THE EXISTENCE OF METRICS OF NONPOSITIVE CURVATURE ON THE BRADY-KRAMMER COMPLEXES FOR FINITE-TYPE ARTIN GROUPS

A Dissertation

by

WOONJUNG CHOI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2004

Major Subject: Mathematics

THE EXISTENCE OF METRICS OF NONPOSITIVE CURVATURE ON THE BRADY-KRAMMER COMPLEXES FOR FINITE-TYPE ARTIN GROUPS

A Dissertation

by

WOONJUNG CHOI

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

Sue Geller (Chair of Committee)	Jon McCammond (Member)
Catherine Yan (Member)	Marcelo Aguiar (Member)
Jianer Chen (Member)	Al Boggess (Head of Department)

May 2004

Major Subject: Mathematics

ABSTRACT

The Existence of Metrics of Nonpositive Curvature on the
Brady-Krammer Complexes for Finite-type Artin Groups. (May 2004)
Woonjung Choi, B.S., Pusan National University;
M.S., Pusan National University
Chair of Advisory Committee: Dr. Sue Geller

My dissertation focuses on the existence of metrics of nonpositive curvature for the simplicial complexes constructed recently by Tom Brady and Daan Krammer for the braid groups and other Artin groups of finite type. In particular, for each Artin group of finite type, there is a recently constructed finite simplicial Eilenberg-Mac Lane space known as its Brady-Krammer complex. The Brady-Krammer complexes are highly symmetric objects.

Prior work on the relationship between the Brady-Krammer complexes and the theory of CAT(0) spaces has produced some positive results in low-dimensions. More specifically, the Brady-Krammer complexes of dimension at most 3 have been shown to support piecewise Euclidean metrics of nonpositive curvature. Similarly, the 4-dimensional Brady-Krammer complexes of type A_4 and type B_4 also support such metrics. In every instance, the metrics assigned respect all of the symmetries alluded to above. The main results of my dissertation show that this pattern does not extend to the Brady-Krammer complexes of type F_4 and D_4. These are the first negative results known about the curvature of these Brady-Krammer complexes. The proofs of my main theorems involve a combination of combinatorial results and computer calculations.

These negative results are particularly striking since Ruth Charney, John Meier

and Kim Whittlesey have shown that a particular complex closely related to each Brady-Krammer complex admits an asymmetric metric satisfying a weak version of nonpositive curvature. Thus, one corollary of my results is that the weak asymmetric version of a CAT(0) metric (initially defined by Mladen Bestvina) is strictly weaker than the traditional version.

To my parents

ACKNOWLEDGMENTS

I am grateful to Dr. Jon McCammond for his patience and support for my slow development in understanding of the subject, for guiding me through my Ph.D. study, and for being a role model as a mentor. Without his help, this dissertation would never have been realized. I also deeply appreciate Dr. Sue Geller's support and very valuable advice during the final semester as an official chair of the committee for my Ph.D. dissertation. Drs. Catherine Yan, Marcelo Aguiar and Jianer Chen provided me with their invaluable comments as well. It would not have been possible to successfully complete my Ph.D. studies without appropriate support from the department of Mathematics and its staff at the Texas A&M University.

I enjoyed many of my colleagues at the department, especially Mr. Dukjin Nam and Mr. Beng Seong Ong for their willingness to help others, including myself, on various occasions.

I would like to express my sincere appreciation to my friend, Seungchan Kim, for his support, encouragement and advice, including editorial comments.

All of my family, parents, brother, and sister and her family, have consistently encouraged and supported me to pursue and finish Ph.D. study and I could have not done it without their support.

Lastly, I thank God for giving me the strength and talent to make this happen.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

Because almost all questions regarding finitely presented groups are known to be undecidable in general, researchers of infinite groups have long focused on those special classes of groups where everything seems to work: free groups and their automorphisms, surface groups, one-relator groups, Coxeter groups, Artin groups, groups which satisfy some sort of small-cancellation conditions, etc. Beginning in the early 1980s M. Gromov strongly advocated the viewpoint that every finitely generated group has an intrinsic geometry which has a major impact on its algebraic structure, and over the past twenty years it has become clear that many of the standard classes of groups are negatively or nonpositively curved in a precise geometric sense [36, 37]. Thus nonpositive curvature is now seen as a partial explanation of the exceptional nature of these classes of groups.

Building on Gromov's work (and the work of other differential geometers) many theories of curvature have been developed which are closely related to infinite group theory [36, 16, 33]. Because there are many different competing theories, there has been a major effort in recent years to see which theories of curvature can be applied to each of the standard classes of groups studied by geometric group theorists and, in particular, to see if the various theories can be distinguished based on these investigations.

The focus of my dissertation is on one of the standard classes of groups (Artin groups of finite-type) and one of the standard theories of nonpositive curvature as developed by Cartan, Alexandrov and Topogonov (CAT(0) spaces and groups, named

---

The journal model is the Bulletin of the American Mathematical Society.

in honor of these three early researchers in the area). Basic results about Artin groups and CAT(0) spaces are reviewed in Chapter II.

For each Artin group of finite type, there is a recently constructed finite simplicial Eilenberg-Mac Lane space known as its Brady-Krammer complex. The Brady-Krammer complexes are highly symmetric objects. In particular, the universal cover of the Brady-Krammer complex has several symmetries in addition to those required by the group structure (e.g., the deck transformations of the covering map). There is always a natural simplicial isometry which I call the *vertical translation map*, as well as additional symmetries which arise as a result of the symmetries of the Dynkin diagram used to define the group. Of particular importance is the simplicial complex obtained by "quotienting" the universal cover by the vertical translation map. I call this quotient the universal cover of the *cross-section complex*. The construction of these spaces and a precise description of these additional symmetries will be described in Chapter III.

Prior work on the relationship between the Brady-Krammer complexes and the theory of CAT(0) spaces has produced some positive results in low-dimensions. More specifically, the Brady-Krammer complexes of dimension at most 3 have been shown to support piecewise Euclidean metrics of nonpositive curvature. Similarly, the 4-dimensional Brady-Krammer complexes of type A_4 and type B_4 also support such metrics. In every instance, the metrics assigned respect all of the symmetries alluded to above. The main results of my dissertation show that this pattern does not extend to the Brady-Krammer complexes of type F_4 and D_4. These are the first negative results along these lines.

Theorem A (Type F_4). *The Brady-Krammer complex for the Artin group of type F_4 does not support a piecewise Euclidean metric of nonpositive curvature which respects*

all of the symmetries of the complex.

More precisely, if a piecewise Euclidean metric is chosen so that the vertical translation map is an isometry of the universal cover and the symmetries of the Dynkin diagram also induce isometries of the universal cover, then the complex with this metric is not nonpositively curved. A similar result holds for the Brady-Krammer complex of type D_4.

Theorem B (Type D_4). *The Brady-Krammer complex for the Artin group of type D_4 does not support a piecewise Euclidean metric of nonpositive curvature which respects all of the symmetries of the complex.*

These negative results are particularly striking since Ruth Charney, John Meier and Kim Whittlesey have shown that the cross-section complex of each Brady-Krammer complex admits an asymmetric metric satisfying a weak version of nonpositive curvature. Thus, one corollary of my results is that the weak asymmetric version of a CAT(0) metric (initially defined by Mladen Bestvina) is strictly weaker than the traditional version.

Theorem C (Weak CAT(0)). *The existence of a* CAT(0) *metric is strictly stronger than the existence of a weak asymmetric* CAT(0) *metric in the following sense. There exists a simplicial complex L that admits a weak asymmetric* CAT(0) *metric invariant under all of the orientation-preserving symmetries of the complex, but L does not admit a* CAT(0) *metric invariant under all of its symmetries.*

Theorem A is proven in Chapter IV and Theorems B and C are proven in Chapter V. The proofs of these theorems involve a large number of calculations completed with the aid of a computer. The routines I wrote to perform these computations are listed in Appendix A. The current state of knowledge about the relationship between

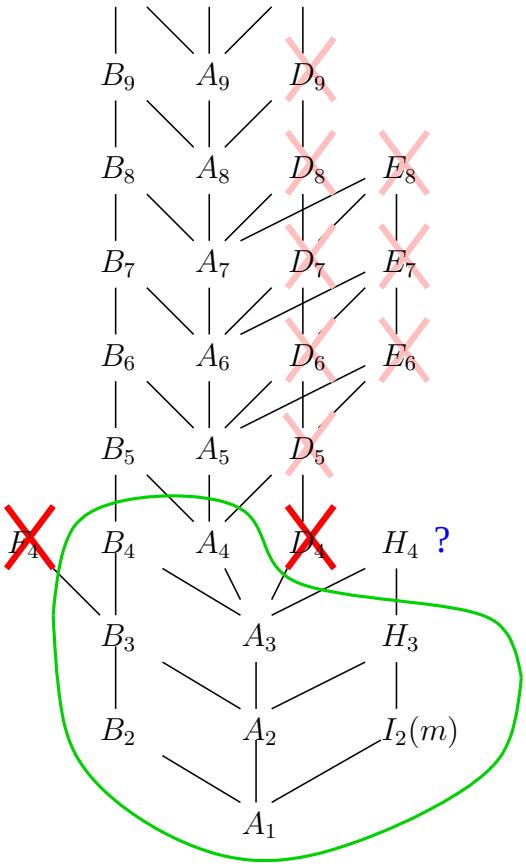


Fig. 1. A summary of curvature results for Artin groups of finite type

the Brady-Krammer complexes and metrics of nonpositive curvature is summarized in Figure 1. This figure is discussed in more detail in the final chapter, Chapter VI, where I summarize my findings.

CHAPTER II

ARTIN GROUPS AND CAT(0) SPACES

As mentioned in the introduction, my dissertation focuses on the class of Artin groups of finite-type and the theory of CAT(0) spaces. In this chapter I review the basic definitions and results in these areas which are needed in the later chapters.

A. Artin groups

Coxeter groups and Artin groups are two classes of groups defined by finite presentations of a very special form. This special form is derived from the standard presentations for the finite reflection groups (in the case of Coxeter groups) and from Artin's presentations for the braid groups (in the case of Artin groups). Both classes of groups are well-behaved enough to prove strong results, but complicated enough to produce numerous counterexamples to long-standing conjectures. As a result, they are an important testing ground for the various general theories of nonpositive curvature within geometric group theory.

As mentioned above Coxeter groups and Artin groups are two closely connected classes of groups which generalize the symmetric groups and the braid groups, respectively. In this section, I review the definitions of all four of these classes of groups and describe their basic properties. I begin with the symmetric groups and the braid groups.

Definition A.1 (Symmetric groups). Recall that the symmetric group Σ_n is the group of bijections from the set $[n] = \{1, 2, \ldots, n\}$ to itself under function composition. This group is generated by the adjacent transpositions σ_i, $i \in [n-1]$, which switch i and $i+1$ fixing all other elements. Perhaps surprisingly, there is a presenta-

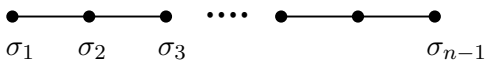


Fig. 2. The Dynkin diagram for Σ_n

tion for Σ_n generated by the σ_i whose only relations record the order of the products $\sigma_i\sigma_j$. This order is 1 when $i = j$, 3 when $|i - j| = 1$, and 2 otherwise. Because each generator has order 2, the presentation of Σ_n can be rewritten in the following form.

$$\Sigma_n = \left\langle \sigma_1, \ldots, \sigma_{n-1} : \begin{array}{rl} \sigma_i^2 = 1 & \text{for all } i \\ \sigma_i\sigma_j = \sigma_j\sigma_i & \text{if } |i-j| > 1 \\ \sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j & \text{if } |i-j| = 1 \end{array} \right\rangle$$

Conventionally, this presentation is succinctly summarized in a labeled graph known as a Dynkin diagram. This graph has one vertex for each generator and an edge connecting a pair of distinct vertices if and only if the product of the corresponding generators has order 3. If two vertices are not connected by an edge then the generators to which they correspond commute, and every generator is assumed to have order 2. Using these conventions, the presentation for Σ_n is encoded in the graph shown in Figure 2.

Definition A.2 (Braid groups). The n-string braid group, $\textsc{Braid}_n$, is the group of motions of n distinct points in the unit disc $\mathbf{D}^2$ up to isotopy. To convert this description to the more familiar picture of physical braids in $\mathbf{R}^3$ one can consider the trails left by these n points over time in the cylinder $\mathbf{D}^2 \times [0, 1]$ where the points are required to start and end in some specified configuration. Figure 3 shows an element of $\textsc{Braid}_4$. Multiplication consists of stacking pictures of this type. This group was first defined by Emil Artin in 1926 [1] in a paper where he proved that $\textsc{Braid}_n$ has

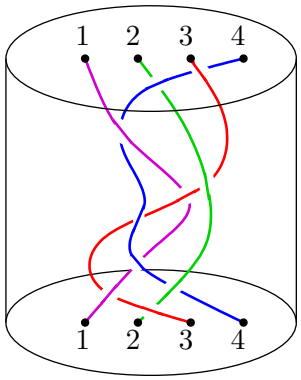


Fig. 3. A typical element in $\textsc{Braid}_4$

the following presentation.

$$\textsc{Braid}_n = \left\langle \sigma_1, \ldots, \sigma_{n-1} : \begin{array}{ll} \sigma_i\sigma_j = \sigma_j\sigma_i & \text{if } |i-j| > 1 \\ \sigma_i\sigma_j\sigma_i = \sigma_j\sigma_i\sigma_j & \text{if } |i-j| = 1 \end{array} \right\rangle$$

In this presentation, the strands are numbered 1 through n and the element σ_i is the motion which crosses the i^{th} strand in front of the $i+1^{st}$ strand leaving all others fixed. The similarity with the presention of symmetric group Σ_n is clear. In particular, there is a natural group homomorphism $\textsc{Braid}_n \to \Sigma_n$ which sends the generator σ_i of $\textsc{Braid}_n$ to the generator σ_i of Σ_n. Notice also that the presentation of the braid group can be summarized by the exact same Dynkin diagram only without the convention that the square of each generator is trivial.

Emil Artin studied this presentation of the braid groups on a number of occasions [2, 3]. His work, and the connection between Σ_n and $\textsc{Braid}_n$ via their Dynkin diagrams, was the motivation for the definition of an arbitrary Artin group. Let Γ be a finite graph with each edge labeled by an integer greater than 2 or by ∞, and let $(a,b)^n$ denote the first n letters of $(ab)^n$.

Definition A.3 (Artin groups and Coxeter groups). The *Artin group* A_Γ is the group generated by a set in one to one correspondence with the vertices of Γ together with a relation of the form $(a,b)^n = (b,a)^n$ whenever the vertices corresponding to a and b are joined by an edge labeled n. When the label is ∞, no relation is added to the presentation. In addition the relation $ab = ba$ is added when the vertices labeled a and b are not connected by an edge. To simplify the most common Dynkin diagrams the labels equal to 3 are usually suppressed. The *Coxeter group* W_Γ has the same presentation as the Artin group A_Γ with the additional relations $a^2 = 1$ for each generator a. Technically speaking, a Coxeter group is any group W which has a Coxeter presentation with respect to some generating set S. Notice that the presentation can be completely recovered from the generating set alone by recording the orders of the products of pairs of generators. The ordered pair (W,S) is called a *Coxeter system* and S is called a set of *Coxeter generators*. Similarly, the ordered pair (A,S) is called an *Artin system* and S is a set of *Artin generators*. In either context, S is called a *standard generating set* for W or A.

The standard reference works on Coxeter groups are Bourbaki [6], Humphreys [39], Grove-Benson [38] and Kane [40]. Artin groups were defined in 1972 by Brieskorn and Saito [17] and Deligne [29], independently. They are well-established in the journal literature [13, 23, 22, 27, 35, 20, 30, 18, 19, 7, 12, 8, 44, 45, 25, 26] but at present no book-length studies of Artin groups have been published.

Remark A.4 (An alternative convention). There is an alternative convention for associating labeled graphs with presentations which is common in the general study of Artin groups. Edges labeled 2 are included, edges labeled ∞ are omitted, and no labels are suppressed. This convention has the advantage that the edges drawn correspond exactly to the relations in the presentation between pairs of generators.

The standard convention, on the other hand, greatly simplifies the diagrams for the finite Coxeter groups.

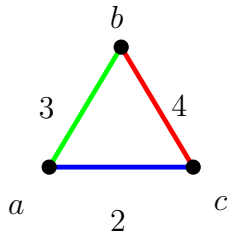


Fig. 4. A Dynkin diagram using the alternative convention

Example A.5. Let Γ be the labeled graph shown in Figure 4 using the alternative convention. The presentation of the Artin group A_Γ is

$$\langle a, b, c |\ aba = bab, ac = ca, bcbc = cbcb \rangle$$

and the presentation of the Coxeter group W_Γ is

$$\langle a, b, c |\ aba = bab, ac = ca, bcbc = cbcb, a^2 = b^2 = c^2 = 1 \rangle .$$

The main motivation for introducing the class of Coxeter groups arose from the classification of finite groups acting on Euclidean space which are generated by reflections. A *reflection* is a linear transformation of $\mathbf{R}^n$ which fixes an $(n-1)$-dimensional subspace and sends each vector which is orthogonal to the subspace to its negative. The class of finite reflection groups corresponds exactly to the class of finite Coxeter groups.

A finite Coxeter group with a disconnected Dynkin diagram splits as a direct product of groups and the corresponding finite reflection group splits in a similar way as a direct product of finite reflection groups on spaces of smaller dimension. Thus it is

often sufficient to study those finite Coxeter groups with connected Dynkin diagrams. These groups are called *irreducible*. The irreducible finite Coxeter groups have been classified and their Dynkin diagrams are shown in Figure 5. The names associated with the finite irreducible Coxeter groups, also known as the Cartan-Killing type, are A_n $(n \geq 1)$, B_n $(n \geq 2)$, D_n $(n \geq 4)$, E_6, E_7, E_8, F_4, H_3, H_4 and $I_2(m)$ $(m \geq 2)$. In this classification the symmetric group Σ_n corresponds to type A_{n-1}. An Artin group defined by the Dynkin diagram of a finite Coxeter group is called a *finite-type* Artin group. Although all of these groups are infinite, their structure is significantly simpler than the structure of an arbitrary Artin group.

The relationships among the six classes of groups defined above are shown schematically in Figure 6, where the vertical arrows allude to the surjective group homomorphisms which result when the square of each generator is set to 1.

Definition A.6 (Reflections and Coxeter elements). Let (W, S) be a finite Coxeter system. Viewed as a finite reflection group, it is clear that the set of all reflections T is a union of conjugacy classes in W since the conjugate of a reflection is a reflection. Notice that each Coxeter generating set S is a subset of T and that several different subsets of T form Coxeter generating sets. The set of all reflections is used in the construction of the Brady-Krammer complexes. Another important class of elements inside W is the collection of Coxeter elements. A *Coxeter element* is a product of all of the generators in some Coxeter generating set S in some order. For example, in Σ_n, the element $\sigma_1\sigma_2\cdots\sigma_{n-1}$ is a Coxeter element and the collection of Coxeter elements is exactly the collection of permutations represented by n-cycles. This illustrates a general fact which we record for later use.

Theorem A.7 (Coxeter elements are conjugate). *In a finite Coxeter group W, all Coxeter elements are conjugate.*

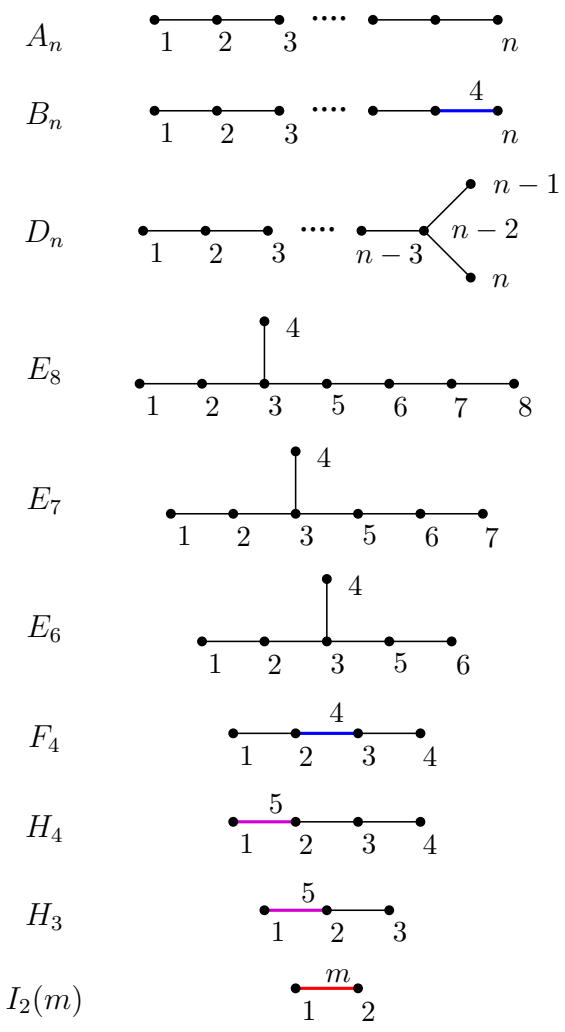


Fig. 5. The Dynkin diagrams for the irreducible finite Coxeter groups

$$\begin{array}{ccccc} \{\text{Braid groups}\} & \subset & \{\text{Finite-type Artin groups}\} & \subset & \{\text{Artin groups}\} \\ \downarrow & & \downarrow & & \downarrow \\ \{\text{Symmetric groups}\} & \subset & \{\text{Finite Coxeter groups}\} & \subset & \{\text{Coxeter groups}\} \end{array}$$

Fig. 6. Six classes of groups and their interrelations

B. CAT(0) spaces

Every finitely presented group G is the fundamental group of a compact path-connected metric space X. The geometric properties of the space X, such as its curvature, can have a profound influence on the algebraic structure of G. For example, if X has the property of being locally CAT(0) (defined in detail below), then X is an Eilenberg-Mac Lane space for G, the geometric dimension of X provides an upper bound on the cohomological dimension of G, and as a result, G must be torsion-free. Over the past decade or so, the theory of CAT(0) spaces has been an active area of research within geometric group theory. A standard reference for this theory is the book by Bridson and Haefliger [16]. I begin by giving a precise definition of what it means for a space to be CAT(0). The first requirement is that the metric space under consideration be a geodesic metric space.

Definition B.1 (Geodesic metric space). Let (X, d) be a metric space. A *geodesic path* is an isometric embedding of a closed interval of the reals into X. More explicitly, a geodesic path joining $x \in X$ to $y \in X$ (or more briefly a *geodesic* from x to y) is a map c from a closed interval $[0, l] \subset \mathbf{R}$ to X such that $c(0) = x$, $c(l) = y$ and $d(c(t), c(t')) = |t - t'|$ for all $t, t' \in [0, l]$. In particular $l = d(x, y)$. A *closed geodesic loop* is an isometric embedding of a metric circle. The metric space (X, d) is said to be a *geodesic metric space* if every two points in X are joined by a geodesic.

Associated to any geodesic metric space are those groups which act on it geo-

metrically.

Definition B.2 (Geometric actions). Let G be a group acting by isometries on a metric space X. Recall that an *isometry* is a bijective map between two metric spaces that preserves distances. The action is said to be *proper* (alternatively, "G acts properly on X") if for each $x \in X$ there exists a number $r > 0$ such that there are only finite number of elements in G which fail to send x outside the ball of radius r around x. It is said to be *cocompact* if there exists a compact set $K \subseteq X$ such that the translates of K under the action of G cover X. If an action by isometries is both proper and cocompact, then G is said to act *geometrically* on X.

The final definition we need is that of a comparison triangle.

Definition B.3 (Comparison triangles). Let X be a geodesic metric space and let x, y and z be points in X. Since the distances between these points satisfy the triangle inequality, there exists a (possibly degenerate) triangle in $\mathbf{R}^2$ with vertices x', y' and z' so that the lengths of the sides of this triangle agree with the distances between the corresponding points in X. This triangle, which is unique up to an isometry of $\mathbf{R}^2$, is called the *comparison triangle*. Notice that if we pick a geodesic path from x to y then we can use the metric of X to map the points along this path bijectively to the points on the line segment connectiong x' and y'.

Definition B.4 (CAT(0) spaces and groups). A geodesic metric space X is CAT(0) if each triangle of geodesics in X is thinner than the corresponding triangle in the Euclidean plane. More precisely, for all points $x, y, z \in X$, and for all choices of geodesic paths connecting x to y, y to z and x to z, and for any point p on the chosen geodesic connecting x and y, the distance between p and z in X should be at most the distance between p' and z' in the comparison triangle X'. See Fig. 7 for an illustration.

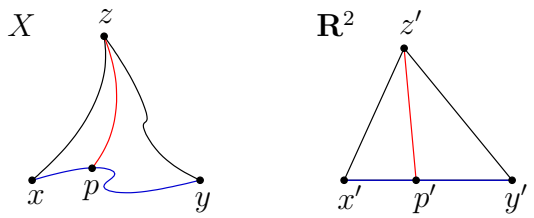


Fig. 7. A geodesic triangle and the comparison triangle

A group which acts geometrically (i.e. properly cocompactly by isometries) on a CAT(0) metric space is called a CAT(0) *group*. A space whose universal cover is CAT(0) is called a *locally* CAT(0) space. Locally CAT(0) spaces are also sometimes referred to as *nonpositively curved spaces*. Notice that if X is a compact nonpositively curved space then its fundamental group automatically acts geometrically on the universal cover of X so that $\pi_1(X)$ is a CAT(0) group.

One of the easiest ways to construct a CAT(0) metric space is by adding a piecewise Euclidean metric to a simplicial complex. The main ideas are sketched below. See [31] for more detail.

Definition B.5 (Piecewise Euclidean complexes). A *piecewise Euclidean complex* X is a simplicial complex in which each simplex is given a Euclidean metric and the induced metrics on the intersections always agree.

Determining whether a piecewise Euclidean complex is nonpositively curved can be reformulated as a condition on the links of cells.

Definition B.6 (Links). Let X be a piecewise Euclidean complex and x be a point in X. The *link of x in X (usually written $lk(x, X)$)* is the set of unit tangent vectors to X at x. The *link of B in X ($lk(B, X)$)* is the set of unit tangent vectors to x in X which are orthogonal to B for any x in the interior of B. Notice that $lk(B, X)$ does

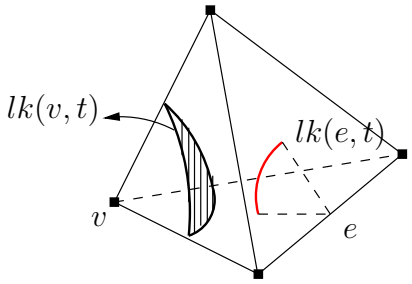


Fig. 8. Examples of links

not depend on the choice of x. These definitions are easiest to understand through examples. Consider the solid tetrahedron t shown in Figure 8. The link of the vertex v in t is the spherical triangle (in the unit sphere) labeled $lk(v,t)$. If x is a point in the edge e then $lk(x,t)$ is a spherical lune while $lk(e,t)$ is the short spherical arc shown whose length is the size of the dihedral angle.

Theorem B.7 (Gromov's link condition). *A piecewise Euclidean complex is nonpositively curved if and only if the link of each cell does not contain a closed geodesic loop of length less than* 2π.

For example, a solid tetrahedron in $\mathbf{R}^3$ is nonpositively curved, but its boundary is not. In high dimensions, testing for closed, short geodesic loops is complicated, but in dimension 3, Murray Elder and Jon McCammond have developed procedures to carry out this type of test [32].

CHAPTER III

BRADY-KRAMMER COMPLEXES

For each finite-type Artin group A_Γ there is a finite simplicial Eilenberg-Mac Lane space K_Γ, called its Brady-Krammer complex, which has A_Γ as its fundamental group. These complexes were first constructed for the braid groups independently by Tom Brady [11] and Daan Krammer [41, 42]. This construction was subsequently generalized to arbitrary Artin groups of finite-type by David Bessis [4] and by Tom Brady and Colum Watt [9]. In this chapter, I review the construction of these complexes, describe their symmetries, and discuss the previously known curvature results for the Brady-Krammer complexes.

A. Constructing the Brady-Krammer complexes

Let A denote one of the Artin groups of finite-type derived from a Dynkin diagram Γ, and let W be the corresponding Coxeter group derived from the same diagram. In both cases, denote the standard generating set by S. The Brady-Krammer complex for A is constructed through a series of steps. I start with an overview of the construction followed by a more detailed description of each step.

1. First, construct the Cayley graph of W with respect to the set of all of its reflections T.

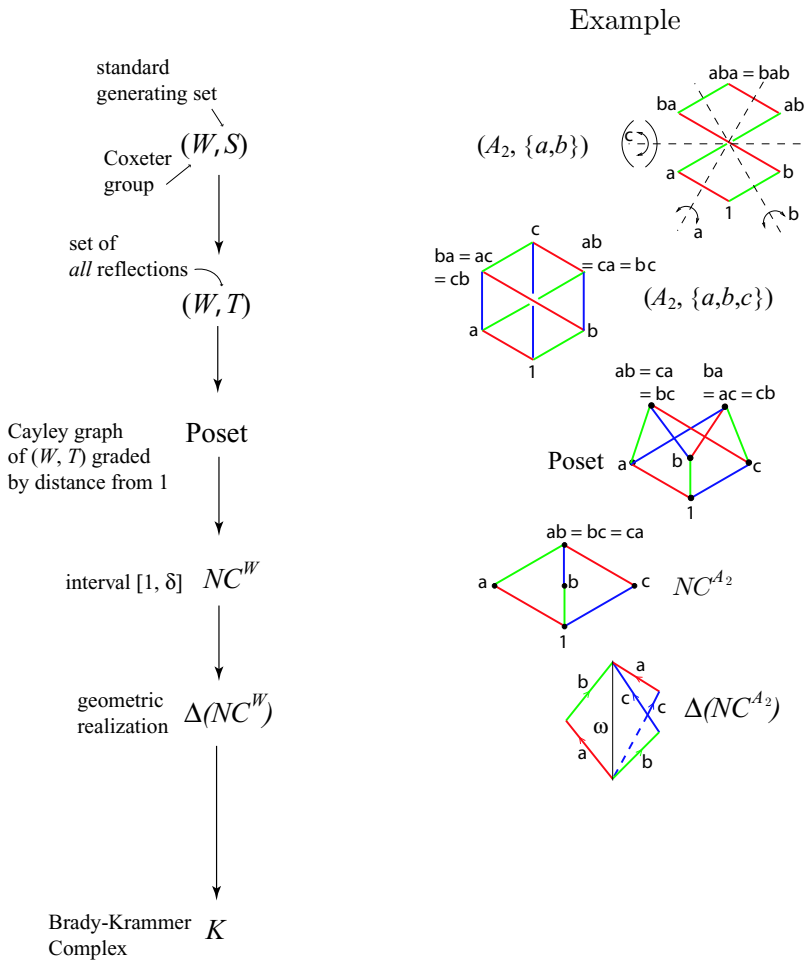
2. Next, orient this Cayley graph based on the distance from the identity vertex to get a partially ordered set. The noncrossing partitions in W will be the interval in this poset between the identity vertex and the vertex labeled by a Coxeter element. In particular, we select the subgraph of paths from 1 to a specific Coxeter element as our interval.

3. Finally, consider the geometric realization of the poset of noncrossing partitions in W. Certain simplices in this geometric realization are identified based on the edge labels of the underlying Cayley graph to produce the final complex.

As I describe each of these three steps in detail, I will use the Coxeter and Artin groups associated to the A_2 Dynkin diagram to illustrate the construction. Figure 9 illustrates the procedure.

Step 1. For each finite Coxeter group W, the set of all reflections T was defined in Definition A.6. Recall that the (right) Cayley graph $\textsc{Cayley}(G, X)$ of a group G with respect to a generating set X is the directed graph whose vertices are labeled by elements of G and whose edges are in one to one correspondence with the set $G \times X$. In particular, the directed edge (g, x) starts at the vertex labeled g and ends at the vertex labeled by $g \cdot x$. Since the first coordinate of this ordered pair merely records the vertex at which it starts, the *label* of this edge is defined to be x. The fact that X is a generating set implies that $\textsc{Cayley}(G, X)$ is connected. In the case of a Coxeter group, all of the generators have order 2 so the Cayley graph can be simplified by replacing each pair of oriented edges with same label connecting the same vertices with a single unoriented edge bearing the common label. In the A_2 Coxeter group, i.e. Σ_3, the standard generating set consists of two reflections, a and b. See the left hand side of Figure 10. The set of all reflections contains a third element c. The simplified Cayley graph of Σ_3 with respect to all three reflections is shown on the right hand side of Figure 10.

Step 2. For each vertex in $\textsc{Cayley}(W, T)$ we associate a *height* which records its combinatorial distance from the identity vertex. Geometrically the height of an element is $n-k$ where n is the number of vertices in the Dynkin diagram for W (which is



$$w = \text{Coxeter element} = \prod_{s \in S} s$$

Fig. 9. Constructing Brady-Krammer complexes

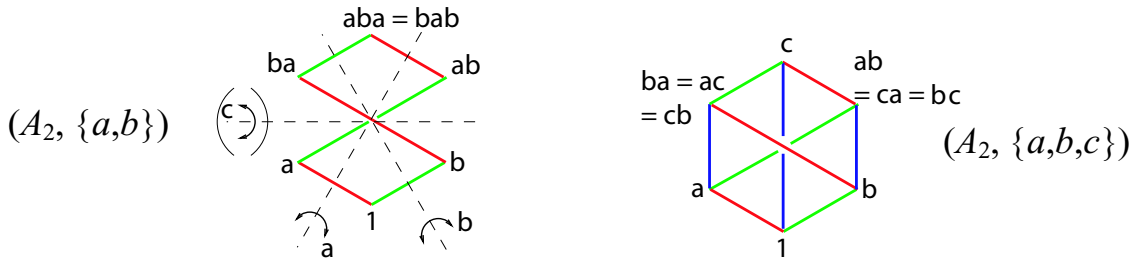


Fig. 10. The Cayley graphs of Σ_3 with respect to the standard generating set and and all reflections

also the dimension of the Euclidean space on which W acts as a reflection group) and k is the dimension of the subspace fixed under the action of this group element. Due to the structure of Coxeter groups, each edge in the Cayley graph connects vertices with distinct (but adjacent) heights. If we view the endpoint closer to the identity as being "below" the other endpoint, then this converts the Cayley graph into a Hasse diagram for a poset. Recall that the *Hasse diagram* of a poset P is the graph whose vertices are the elements of P and whose edges are the covering relations (i.e. those inequalities $x < y$ where no other element is strictly between x and y). Algebraically $g \leq h$ in this ordering if and only if there is a minimal factorization of h into a product of reflections such that g is the product of an initial segment of this factorization. Let w be one of the Coxeter elements of W and let NC^W be the interval in this poset between the identity vertex and the vertex labeled w. Because all Coxeter elements are conjugate (Theorem A.7) and because W acts on the Cayley graph by conjugation, the structure of this interval is independent of the Coxeter element chosen. The poset corresponding to this interval, called the *noncrossing partitions* of type W, is known to be a graded bounded self-dual (and locally self-dual) lattice. Recall that a poset is *bounded* if it has a unique maximum element and a unique minimum element

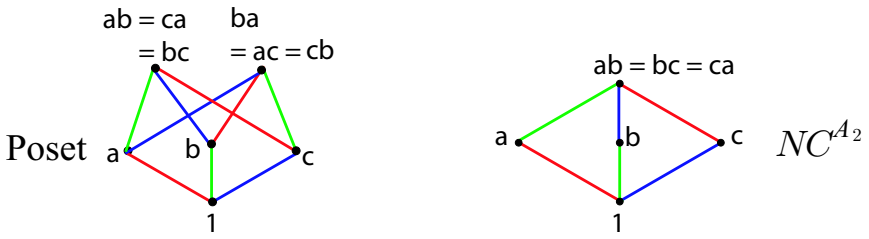


Fig. 11. The full poset and the noncrossing partition lattice for Σ_3

and it is *graded* if all maximal chains have the same length. It is a *lattice* if every pair of elements has a unique least upper bound and a unique greatest lower bound. Finally a poset is *self-dual* if there is an order-reversing bijection from it to itself (and locally self-dual if every interval is self-dual). The grading of the poset is provided by the height function. It is bounded because the vertices labeled 1 and w are clearly the unique minimum and maximum elements respectively. The fact that all of these posets are self-dual lattices is not as immediate. See [4] for details. In the case of Σ_3 the full poset and its restriction to the noncrossing partition lattice are shown in the Figure 11.

The reason these lattices are called noncrossing partition lattices is that this procedure in the case of the symmetric groups produces the traditional noncrossing partition lattices studied by combinatorialists. Since traditional noncrossing partitions are not needed later in this dissertation I will not define them here, but some idea of their structure can be seen in Figure 12. See [43] and the references therein for more details.

Step 3. The next step in the construction is to replace NC^W with its geometric realization. See Figure 13. Recall that the *geometric realization* of a partially ordered set P is a simplicial complex whose vertices correspond to elements of P and whose

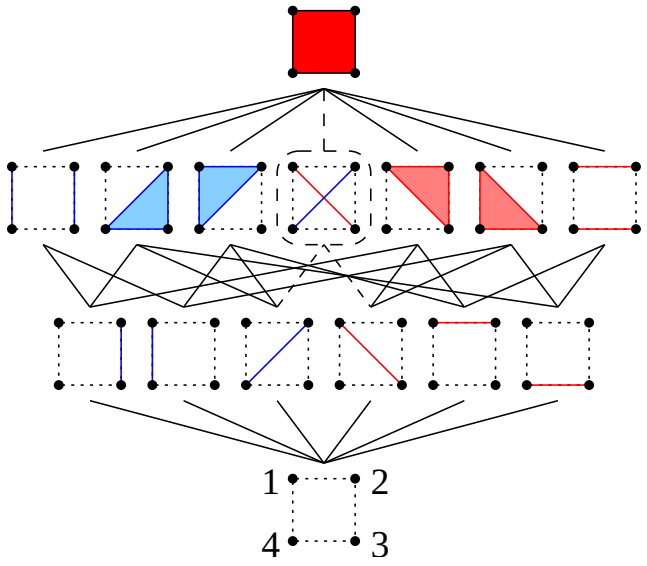


Fig. 12. The (traditional) noncrossing partition lattice for $n = 4$

$k-$simplicies are the strictly increasing sequences $g_0 < \cdots < g_k$ of elements of P. This geometric realization is often denoted $\Delta(P)$. Since the 1-cells of $\Delta(NC^W)$ correspond to pairs of group elements with $g < h$, we can think of this edge as being assigned a label $g^{-1}h$. With this labeling scheme, the edges of $\Delta(NC^W)$ which correspond to the edges of the Hasse diagram of NC^W are labeled by elements of T, while the other edges are labeled by more complicated group elements (i.e. group elements of height greater than one and lower dimensional fixed sets). In addition to the labels, edges in the geometric realization come equipped with a natural orientation coming from the poset ordering. The final operation involves identifying two simplices in $\Delta(NC^W)$ if and only if their one-skeleta have identical labels. The particular identification of these simplices we use is the unique identification which preserves labels and orientations. The resulting complex K is called the *Brady-Krammer complex* for A. Under the sequence of identifications the final complex always has a unique vertex.

Notice that even though every step of the construction uses the Coxeter group W rather than the corresponding Artin group A, the final complex is called the Brady-Krammer complex for A precisely because A happens to be the fundamental group of the resulting space. In the case of Σ_3, the geometric realization consists of three triangles sharing a common edge which is labeled by the chosen Coxeter element. There are three identifications which need to be made since there are two 1-cells labeled a, two 1-cells labeled b and two 1-cells labeled c. Since no two triangles have identical labels in this example, there are no 2-simplices which need to be identified. The fundamental group of the resulting complex, which has one vertex, four edges, and three triangles, is $\textsc{Braid}_3$. The universal cover of this complex is easier to visualize and a portion of it is shown in Figure 14. Topologically the universal cover is homeomorphic to the direct product of an infinite uniformly 3-branching tree and a copy of the reals.

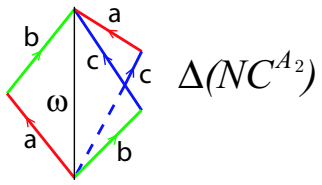


Fig. 13. The geometric realization

Recall that a space X is called an *Eilenberg-Mac Lane space* of type $K(G, n)$, $n \geq 1$, if all of the homotopy groups of X are trivial except for $\pi_n(X)$ which is isomorphic to G. In particular, a reasonably nice space such as a simplicial complex is a $K(G, 1)$ if and only if its fundamental group is G and its universal cover is contractible. As mentioned at the beginning of the chapter, the following results

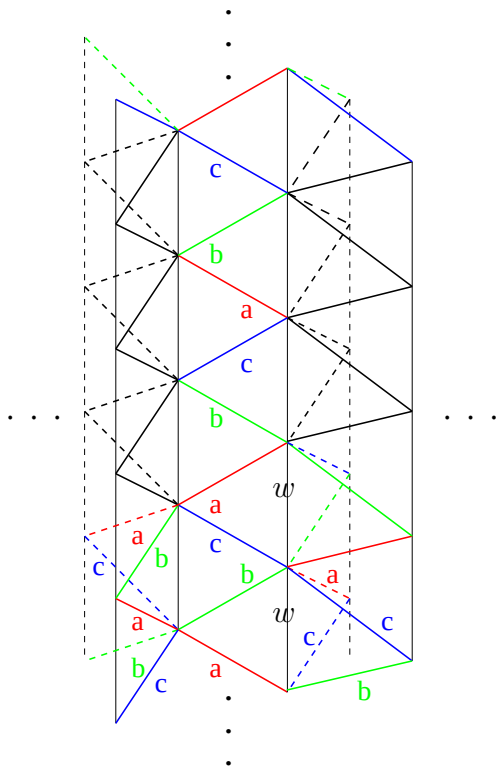


Fig. 14. The universal cover of the Brady-Krammer complex of type A_2

have been established about Brady-Krammer complexes.

Theorem A.1 (T.Brady, D.Krammer). *The Brady-Krammer complex for the braid group* $\textsc{Braid}_n$ *is an Eilenberg-Maclane space of type* $K(\textsc{Braid}_n, 1)$.

Theorem A.2 (D.Bessis, T.Brady-C.Watt). *For each finite-type Artin group* A*, the Brady-Krammer complex* K *for* A *is an Eilenberg-Maclane space of type* $K(A, 1)$.

Recently this construction has been generalized even further to the class of groups with Garside structures [21]. This more general construction is similar to the Brady-Krammer complex in many ways. For example the final result always has a unique vertex. The most important property that it shares with the Brady-Krammer construction is the following.

Theorem A.3 (Charney, Meier and Whittlesey). *Every group G with a Garside structure has a finite Eilenberg-Mac Lane space of type $K(G,1)$ obtained by mimicking the construction of Brady and Krammer.*

Garside structures on groups are an abstraction of the key properties possessed by the noncrossing partitions inside finite-type Artin groups. In 1968 F. Garside gave a new solution to the word problem for the braid groups by showing, in modern terminology, that the braid groups possess a Garside structure. Garside's article was an inspiration for the class of Garside groups defined by P. Dehornoy and L. Paris in [28].

B. Symmetries of the Brady-Krammer complexes

Let K be the Brady-Krammer complex for an Artin group A of finite-type and let $\tilde{K}$ be its universal cover. The complex $\tilde{K}$ has several symmetries in addition to the deck transformations associated with the covering map. This section begins by defining the vertical translation map mentioned in the introduction followed by a detailed discussion of how the vertical translation map decomposes $\tilde{K}$ into columns. Finally the section concludes by discussing a few additional symmetries.

To define the vertical translation map, notice that every vertex in $\tilde{K}$ is the starting point of a unique directed edge labeled by the chosen Coxeter element w. Moreover, the map on vertices which sends each vertex to the endpoint of this unique edge extends to a simplicial isometry of the entire complex $\tilde{K}$. I call this *vertical translation map* t. If we identify the vertices of $\tilde{K}$ with the elements of A then the map t sends the vertex labeled g to the vertex labeled gw. The vertical translation map preserves orientations of edges but the edge labels are not necessarily preserved. More precisely, if e is an edge of $\tilde{K}$ labeled by $x \in W$, then the label on $t(e)$ is the

label of e conjugated by w (in other words, the new label is $x^w := w^{-1}xw$).

The existence of the vertical translation map means that $\tilde{K}$ is a complex composed of columns. A *column* is a smallest full subcomplex of $\tilde{K}$ which contains at least one top dimensional simplex and is invariant under the action of t and t^{-1}. A column in the universal cover of the Brady-Krammer complex of type A_2 is shown in Figure 15.

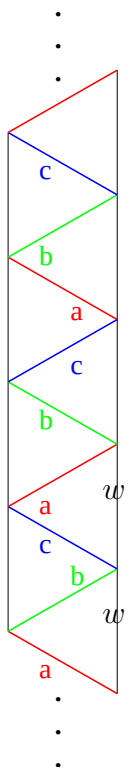


Fig. 15. A column of the universal cover of the Brady-Krammer complex of type A_2

If we restrict our attention to only those edges in a column which are labeled by elements of T, then these edges form a single bi-infinite spiral which winds its way through every vertex in the column. Moreover, the bi-infinite sequence of labels on these edges, $\{x_i\}_{i\in\mathbb{Z}}$, have the property that the product of any n consecutive

labels, in order, is equal to w. To see why this makes sense algebraically, consider the following. A top dimensional simplex in the Brady-Krammer complex corresponds to a maximal chain in the noncrossing partition lattice NC^W, which in turn corresponds to a minimal length factorization of w into reflections. Since a column must contain at least one top dimensional simplex, the edges of the column labeled by elements of T must at least contain a path of length n whose labels correspond to a minimal length factorization of w. Since a column is invariant under the action of t, it must also contain the image of this path under t. Notice that the image of the path starts at the vertex where the original path stopped since the labels of the starting and ending vertices differ by w. This creates a path of length $2n$. Continuing in this way, using t and t^{-1}, we find that the column must contain a bi-infinite path whose edges are labeled by reflections. Finally, notice that the sequence of vertices traversed by this path are invariant under the action of t (and t^{-1}) since the product of every n consecutive labels in this bi-infinite sequence, multiplied in order, is equal to w. For example it is easy to check that if $x_1x_2\cdots x_n$ is a factorization of w then $x_2x_3\cdots x_nx_1^w$ is another factorization of w. Since columns by definition are the minimal full subcomplexes containing at least one top-dimensional simplex invariant under t and t^{-1}, the full subcomplex on this sequence of vertices is a column. An example of 3-dimensional column is shown (horizontally) in Figure 16.

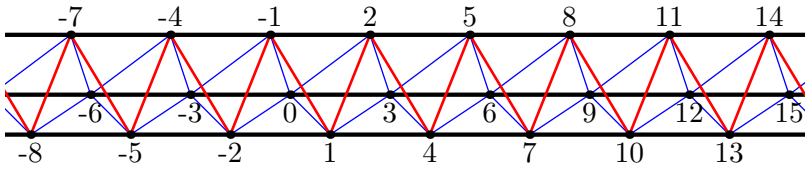


Fig. 16. An example of 3-dimensional column

Topologically each column is homeomorphic to a closed $(n-1)$-ball times the

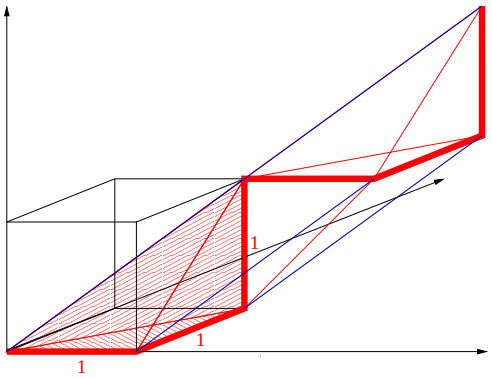


Fig. 17. A column with natural metric embedded in $\mathbf{R}^3$

reals. To see this, identify the vertices to integers according to the order they occur in spiral. Using this identification, two vertices in a column are connected by an edge if and only if their integers differ by at most n where n is the number of vertices in the Dynkin diagram. If we assign each edge a length of $\sqrt{j}$ where j is the difference between the integers assigned to the vertices, then this metric extends to a piecewise Euclidean metric on the entire column. Moreover, with this metric the column is a direct product of a particular Euclidean $(n-1)$-simplex with the reals. This metric is the metric that Tom Brady and Jon McCammond called the *natural metric*. One way to see that this metric on the column really does decompose as a direct product is to embed the metric version of the column into Euclidean n-space. The spiral can be embedded so that each vertex has integer coordinates and each directed edge travels one unit in some positive coordinate direction, cycling through the n possible directions in order. Figure 17 shows this embedding in the case of $n = 3$. Since the distances between nearby vertices in the embedded spiral match the length assigned above, the embedding of the spiral extends to an embedding of the column. Under this embedding all edges labeled w point in the $(1, 1, \ldots, 1)$ direction, and these edges

collect together to form n distinct lines with this direction vector. The embedded column is the convex hull of these n lines, which is clearly invariant under an arbitrary translation in this direction. Thus metrically, the column is a direct product of the real line and the particular $(n-1)$-simplex which results when the column is intersected with a hyperplane perpendicular to the vector $(1, 1, \ldots, 1)$. The metric on this particular $(n-1)$-simplex is not regular, but it is well-known. It arises elsewhere in the study of Coxeter groups and is known as the metric associated with the Coxeter group $\tilde{A}_{n-1}$ [24]. In the case $n = 4$, the metric tetrahedron, after rescaling, has two nonadjacent edges of length 2 and the remaining four edges of length $\sqrt{3}$. The dihedral angles along the edges of length 2 are $\frac{\pi}{2}$ and along the other edges are $\frac{\pi}{3}$. These specific values are needed in the later chapters.

To summarize, we have shown that each column equipped with the natural metric is isometric to a type $\tilde{A}_{n-1}$ simplex cross $\mathbf{R}$. In addition, the overlap between columns is also of the form $\sigma \times \mathbf{R}$ where σ is some subsimplex of the type $\tilde{A}_{n-1}$ simplex. Thus $\tilde{K}$ equipped with the natural metric is isometric to $Y \times \mathbf{R}$ where Y is an $n-1$ dimensional simplicial complex which records the ways the columns intersect. In the example of the Brady-Krammer complex of type A_2, the complex Y is an infinite uniformly three branching tree. By fixing a value of the second coordinate we can find a copy of Y embedded in $\tilde{K}$. For convenience, choose this value so that the subspace includes at least one vertex of $\tilde{K}$. I call this subspace $\tilde{L}$ since it is the universal cover of a subspace L of K which I call the *cross-section complex*. To see that $\tilde{L}$ is simply connected we note that $\tilde{K} = \tilde{L} \times \mathbf{R}$ is simply connected and the fundamental group of a product is the product of the fundamental groups. The image of $\tilde{L}$ under the covering map $\tilde{K} \to K$ is the definition of L. Notice that the simplicial definition of L and $\tilde{L}$ make sense even in the absence of the natural metric. The natural metric is useful because the resulting complex splits as a direct product of two metric spaces

and the vertical translation map and its iterates are isometries which extend to an action of $\mathbf{R}$ on $\tilde{K}$ whose orbit space is exactly $\tilde{L}$. I show in the next chapter that this situation is essentially generic in the sense that any reasonable CAT(0) metric on $\tilde{K}$ will split in this fashion. Finally we note that the fundamental group of L is exactly the subgroup of A which stablizes $\tilde{L}$, i.e. those elements represented by word of exponent sum 0. In the case of the Brady-Krammer complex of type A_2, L has two vertices with three edges connecting them (i.e. a theta graph).

I conclude this section with a brief discussion of additional isometries of $\tilde{K}$. If the Dynkin diagram defining the Artin group A has symmetries, then these symmetries induce symmetries in the noncrossing partition lattice NC^W which, in turn, induce symmetries in the Brady-Krammer complex K, which then induce symmetries in its universal cover $\tilde{K}$. To explain how the symmetries of the Dynkin diagram propagate in this way I need to specify a standard method of selecting a Coxeter element. Since every Dynkin diagram of an irreducible finite Coxeter group is a tree there is a unique well-defined 2-coloring of its vertices. The *standard Coxeter element* in W is a product of its standard reflections so that all of the reflections corresponding to vertices of one color occur before all of the reflections corresponding to vertices of the other color. Since the reflections corresponding to two vertices with the same color commute, the order in which the reflections within a color group are listed is irrelevant. Notice that the only choice made is which color group to list first and that changing this choice merely replaces w with w^{-1}. As a consequence, the image of the standard Coxeter element under the group automorphism corresponding to a diagram symmetry will send w to either w or w^{-1}. Thus the noncrossing partition lattice NC^W is sent to itself by either a lattice automorphism or an order-reversing lattice automorphism (depending on whether the symmetry of the diagram preserves or reverses the splitting of the vertices into two color groups). In either case, the automorphism permutes the labels

but preserves the equivalence classes of edges which have the same label. Because the equivalence classes are preserved, this automorphism becomes an automorphism of its geometric realization $\Delta(NC^W)$ which descends to a label permutting automorphism of K.

The symmetries of K derived from symmetries of Dynkin diagram may or may not correspond to inner automorphisms of W. A second type of isometry is derived from those inner automorphisms of W which fix w. Since it is well-known that the centralizer of a Coxeter element consists solely of powers of that element [39], it is sufficient to consider the inner automorphism which conjugates by w. Conjugation by w induces another lattice automorphism of NC^W which permutes the edge labels but preserves the equivalence classes into which the labels partition the set of edges. Thus we once again have an automorphism of $\Delta(NC^W)$ which descends to a isomorphism of K.

C. Curvature results for the Brady-Krammer complexes

A finite-type Artin group is a good candidate for a CAT(0) group since it is known to have all of the right properties. Moreover, it has been shown that a complex closely associated with these groups satisfies a weak asymmetrical version of the CAT(0) condition. See chapter V for details. In this final section, I review the previously known positive results regarding curvature for the Brady-Krammer complexes. In order to prove that these groups are CAT(0) groups, we need to define a space on which these groups act with a metric that we can show satisfies the CAT(0) condition. We should note that, even once the space with a metric has been defined, it is often nontrivial to check whether the CAT(0) condition is satisfied. A natural candidate for the space is the Brady-Krammer complex, and a candidate for the metric is the

"natural metric" defined on the columns in the previous section. Using the Brady-Krammer complexes with the natural metric, Tom Brady and Jon McCammond have shown the following.

Theorem C.1 (T.Brady-J.McCammond [12, 10, 14]). *The finite-type Artin groups with at most* 3 *generators are* CAT(0) *groups, and the Artin groups* A_4 *and* B_4 *are* CAT(0) *groups.*

One advantage of using the natural metric is that all of the symmetries of the complex are automatically preserved since the same metric is being used for every top dimensional simplex. As a consequence of these prior results, it was natural to conjecture the following.

Conjecture C.2. *The Brady-Krammer complex for each Artin group of finite type supports a nonpositively curved picewise Euclidean metric which respects all of the symmetries of the complex.*

In the next two chapters I show that this conjecture is false.

CHAPTER IV

THE BRADY-KRAMMER COMPLEX OF TYPE F_4

The main goal of this chapter is to prove the following theorem.

Theorem A (Type F_4). *The Brady-Krammer complex for the Artin group of type F_4 does not support a piecewise Euclidean metric of nonpositive curvature which respects all of the symmetries of the complex.*

Throughout this chapter let A denote the Artin group of type F_4: let W denote the Coxeter group of type F_4, and let K denote the Brady-Krammer complex of type F_4 unless explicitly stated otherwise. The Dynkin diagram of type F_4 is shown in Fig. 18. Recall from Chapter III that the complex K is constructed from the geometric realization of a particular finite poset derived from minimum length factorizations of a Coxeter element in W into reflections. The group W has $1,152$ elements and acts as a finite reflection group on $\mathbf{R}^4$. The lattice of noncrossing partitions of type F_4 has 105 elements arranged in 5 levels with 1 element in the first level, 24 elements in the second level, 55 elements in the middle level and symmetrically 24 in the fourth level and 1 in the fifth. To give some idea of its structure, the Hasse diagram of the F_4 poset is shown in Figure 19. The 432 4-simplices in K correspond to the 432 maximal chains in this poset. Since the complex constructed is 4-dimensional and the number of simplices involved is large, the computer is used to facilitate many of the tabulations.

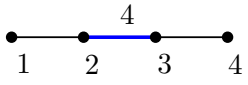


Fig. 18. The Dynkin diagram of type F_4

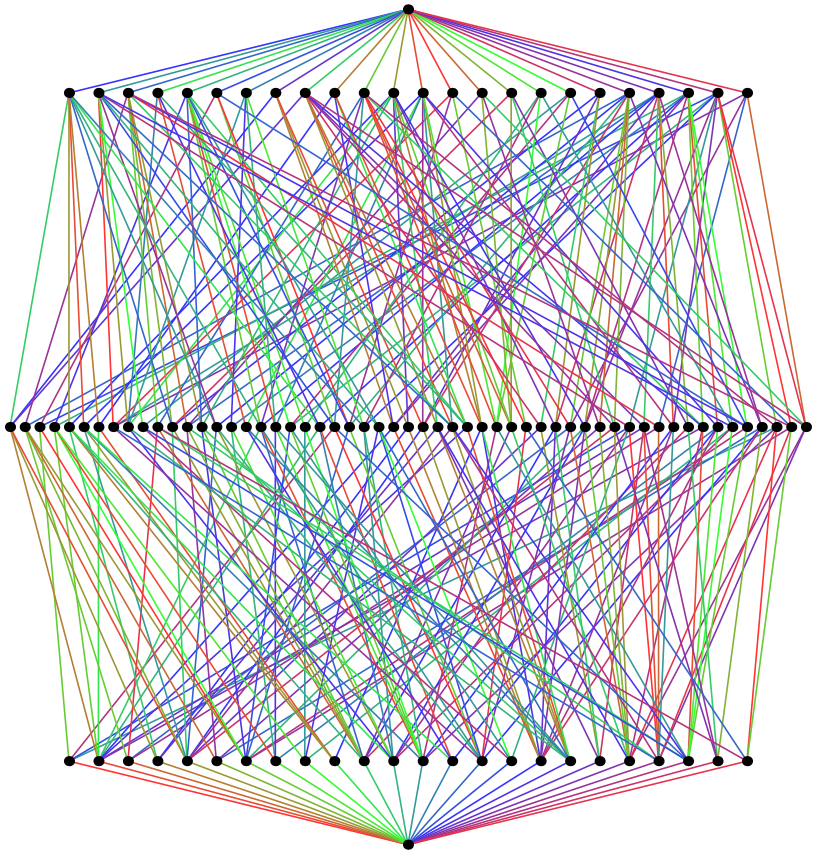


Fig. 19. The lattice of noncrossing partitions of type F_4

The chapter is structured as follows. In Section A, I describe the software routines I wrote and the structure of K in some detail. In Section B, I prove that the Brady-Krammer complex has a piecewise Euclidean metric respecting all of the symmetries of the complex if and only if there exists a piecewise Euclidean metric on the cross-section complex L respecting its symmetries. This has the advantage of replacing a 4-dimensional geometric problem with a 3-dimensional geometric problem where previously developed techniques can be applied. In section C I describe the tetrahedra in the cross-section complex and the possible metrics which would respect its symmetries. In Section D, the links of the edges in the cross-section are examined

and a finite system of linear inequalities is produced which encodes the restrictions on the dihedral angles of the tetrahedra used to construct the cross-section if the metric space under consideration is to be nonpositively curved. In the final section, I analize this system of inequalities and show that none of its solutions lead to a nonpositively curved metric on the cross-section.

A. The structure of the F_4 poset

In this section, the structure of the F_4 poset is investigated using the computer algebra system GAP, which stands for Groups, Algorithms and Programming [34]. The routines I wrote, collected in file called `coxeter.g`, are listed in Appendix A. This section is structured to lead the reader through a typical session using GAP and `coxeter.g`. Note that every GAP instruction ends in a semicolon. After starting GAP the user loads these routines as follows (Figure 20).

```
gap> Read("coxeter.g");
...loading the coxeter reflections

 To begin type something like createCoxeterInfo("A",4);
 where ("A",4) can be replaced with any irreducible Coxeter
 type up to rank 10
```

Fig. 20. GAP Run 1

Following this instruction for type F_4 produces the following output (Figure 21).

```
gap> createCoxeterInfo("F",4);

type = Coxeter group ("F",4)
R    = a list of the 24 reflections of W
w    = coxeter element ([ 1, 3, 2, 4 ])

c  = conjugacy table for R
wc = a list of the conjugates classes for R rel w
```

Fig. 21. GAP Run 2

```
L  = a graded list of permutations below w ([ 1, 24, 55, 24, 1 ])
V  = a list of the 105 permutations listed in L

P  = poset structure for L
PL = poset structure for L with labels

Other information is stored in len, clen, and cw --but
these are mostly for internal use.

  To calculate the column structure, type createColumns();

  Since this group is rank 4, try testGroup(); to see whether
  the Brady-Krammer complex for the Artin group of this type
  is CAT(0) using the standard coxeter shape metric.
```

Fig. 21. Continued

The way GAP represents finite Coxeter groups internally is as a permutation group. In particular, for each genenrator it records the induced permutation of its root system. The *root system* of a finite reflection group acting on $\mathbf{R}^n$ is the collection of unit vectors perpendicular to a hyperplane fixed by one of the reflections. This set of vectors is finite and the action of the Coxeter group on this finite set is faithful. In the case of F_4, there are 24 reflections and thus 48 roots. In the output listed above, the variable R stores the list of 24 reflections of the 48 roots. For example the first element of R is the following permutation (Figure 22).

```
gap> R[1];
(1,25)(2,5)(6,8)(9,11)(10,12)(13,15)(16,18)(23,24)(26,29)
(30,32)(33,35)(34,36)(37,39)(40,42)(47,48)
```

Fig. 22. GAP Run 3

The variable w records the permutation which represents the action of a Coxeter element on the 48 roots. The Coxeter element chosen multiplies the first, third, second, fourth reflections in the list of 24 reflections in this order. These first four reflections correspond to the four vertices of the Dynkin diagram (Figure 23).

```
gap> w;
(1,29,40,38,42,33,25,5,16,14,18,9)(2,20,23,24,22,11,26,44,47,48,46,35)
(3,34,41,45,43,39,27,10,17,21,19,15)(4,6,12,13,8,7,28,30,36,37,32,31)
```

Fig. 23. GAP Run 4

From the permutation, it is clear the order of w is 12. This number is called the *Coxeter number* h. The program has exhaustively searched for all sequences of four reflections whose product is w. The 105 permutations which occur as a product of an initial segment of one of these sequences are listed in V. The variable L records the same 105 permutations but they are separated into groups based on the number of reflections needed to produce them. The program has also compiled a list of the edges of the Hasse diagram. These edges are listed by recording the positions in V of the permutations at either end of the covering relation. Notice that the order that the permutations are listed in V is compatible with the poset order. Thus $V[1]$ is the identity permutation, the elements $V[2]$ through $V[25]$ are the reflections, $V[26]$ through $V[80]$ are the permutations below w which are a product of two reflections, $V[81]$ through $V[104]$ are the permutations below w which are a product of three reflections and $V[105]$ is the permutation w itself. As a consequence all of the ordered pair listed in P contain a smaller number followed by a larger number (Figure 24).

```
gap> P;
[ [ 1, 2 ],[ 1,3 ],[ 1,4 ],[ 1,5 ],[ 1,6 ],[ 1,7 ],[ 1,8 ],
  [ 1,9 ],[ 1,10 ],[ 1,11 ],[ 1,12 ],[ 1,13 ],[ 1,14 ],[ 1,15 ],
  [ 1,16 ],[ 1,17 ],[ 1,18 ],[ 1,19 ],[ 1,20 ],[ 1,21 ],
  [ 1,22 ],[ 1,23 ],[ 1,24 ],[ 1,25 ],[ 2,28 ],[ 2,48 ],
  [ 2,68 ],[ 2,78 ],[ 3,26 ],[ 3,29 ],[ 3,32 ],[ 3,43 ],
  [ 3,47 ],[ 3,52 ],[ 3,57 ],[ 3,69 ],[ 3,73 ],[ 4,27 ],
  [ 4,36 ],[ 4,64 ],[ 4,70 ],[ 5,26 ],[ 5,30 ],[ 5,53 ],
  [ 5,55 ],[ 6,26 ],[ 6,27 ],[ 6,31 ],[ 6,37 ],[ 6,40 ],
  [ 6,54 ],[ 6,56 ],[ 6,62 ],[ 6,67 ],[ 7,27 ],[ 7,38 ],
  [ 7,66 ],[ 7,77 ],[ 8,27 ],[ 8,28 ],[ 8,29 ],[ 8,39 ],
  [ 8,45 ],[ 8,49 ],[ 8,65 ],[ 8,76 ],[ 8,79 ],[ 9,29 ],
  [ 9,41 ],[ 9,44 ],[ 9,74 ],[ 10,29 ],[ 10,46 ],[ 10,51 ],
```

Fig. 24. GAP Run 5

```
[ 10,75 ],[ 11,30 ],[ 11,31 ],[ 11,32 ],[ 11,58 ],[ 12,30 ],
[ 12,36 ],[ 12,37 ],[ 12,38 ],[ 12,39 ],[ 12,43 ],[ 12,60 ],
[ 12,63 ],[ 12,72 ],[ 13,30 ],[ 13,34 ],[ 13,40 ],[ 13,41 ],
[ 13,47 ],[ 13,50 ],[ 13,65 ],[ 13,68 ],[ 13,75 ],[ 14,31 ],
[ 14,43 ],[ 14,44 ],[ 14,45 ],[ 14,46 ],[ 14,50 ],[ 14,59 ],
[ 14,71 ],[ 14,78 ],[ 15,31 ],[ 15,42 ],[ 15,47 ],[ 15,74 ],
[ 16,33 ],[ 16,40 ],[ 16,43 ],[ 16,48 ],[ 16,49 ],[ 16,51 ],
[ 16,61 ],[ 16,64 ],[ 16,77 ],[ 17,35 ],[ 17,45 ],[ 17,51 ],
[ 17,68 ],[ 18,52 ],[ 18,53 ],[ 18,54 ],[ 18,58 ],[ 18,63 ],
[ 18,65 ],[ 18,71 ],[ 18,74 ],[ 18,77 ],[ 19,34 ],[ 19,39 ],
[ 19,42 ],[ 19,44 ],[ 19,51 ],[ 19,55 ],[ 19,56 ],[ 19,57 ],
[ 19,58 ],[ 20,55 ],[ 20,62 ],[ 20,63 ],[ 20,69 ],[ 21,55 ],
[ 21,60 ],[ 21,64 ],[ 21,65 ],[ 21,66 ],[ 21,67 ],[ 21,73 ],
[ 21,78 ],[ 21,80 ],[ 22,33 ],[ 22,38 ],[ 22,56 ],[ 22,68 ],
[ 22,69 ],[ 22,70 ],[ 22,71 ],[ 22,79 ],[ 22,80 ],[ 23,35 ],
[ 23,36 ],[ 23,48 ],[ 23,56 ],[ 23,59 ],[ 23,73 ],[ 23,74 ],
[ 23,75 ],[ 23,76 ],[ 24,67 ],[ 24,69 ],[ 24,72 ],[ 24,77 ],
[ 25,36 ],[ 25,61 ],[ 25,78 ],[ 25,79 ],[ 26,81 ],[ 26,88 ],
[ 26,89 ],[ 27,83 ],[ 27,100 ],[ 28,86 ],[ 28,103 ],[ 29,85 ],
[ 29,102 ],[ 30,81 ],[ 30,92 ],[ 31,81 ],[ 31,93 ],[ 32,81 ],
[ 32,90 ],[ 33,82 ],[ 33,94 ],[ 33,101 ],[ 33,104 ],[ 34,82 ],
[ 34,84 ],[ 34,87 ],[ 34,92 ],[ 35,82 ],[ 35,98 ],[ 36,83 ],
[ 36,91 ],[ 37,81 ],[ 37,83 ],[ 37,96 ],[ 37,99 ],[ 38,83 ],
[ 38,95 ],[ 38,101 ],[ 39,83 ],[ 39,85 ],[ 39,92 ],[ 40,81 ],
[ 40,82 ],[ 40,100 ],[ 41,87 ],[ 41,102 ],[ 42,84 ],[ 42,93 ],
[ 43,81 ],[ 43,85 ],[ 43,91 ],[ 43,101 ],[ 44,85 ],[ 44,87 ],
[ 44,93 ],[ 45,85 ],[ 45,98 ],[ 45,103 ],[ 46,85 ],[ 46,97 ],
[ 47,81 ],[ 47,84 ],[ 47,102 ],[ 48,82 ],[ 48,86 ],[ 48,91 ],
[ 49,85 ],[ 49,86 ],[ 49,94 ],[ 49,100 ],[ 50,81 ],[ 50,87 ],
[ 50,97 ],[ 50,103 ],[ 51,82 ],[ 51,85 ],[ 52,88 ],[ 52,90 ],
[ 52,101 ],[ 52,102 ],[ 53,88 ],[ 53,92 ],[ 54,88 ],[ 54,93 ],
[ 54,99 ],[ 54,100 ],[ 55,89 ],[ 55,92 ],[ 56,82 ],[ 56,83 ],
[ 56,89 ],[ 56,93 ],[ 57,84 ],[ 57,85 ],[ 57,89 ],[ 57,90 ],
[ 58,90 ],[ 58,92 ],[ 58,93 ],[ 59,91 ],[ 59,93 ],[ 59,97 ],
[ 59,98 ],[ 60,91 ],[ 60,92 ],[ 60,95 ],[ 60,96 ],[ 61,91 ],
[ 61,94 ],[ 62,89 ],[ 62,99 ],[ 63,92 ],[ 63,99 ],[ 63,101 ],
[ 64,91 ],[ 64,100 ],[ 64,104 ],[ 65,92 ],[ 65,100 ],[ 65,102 ],
[ 65,103 ],[ 66,95 ],[ 66,100 ],[ 67,89 ],[ 67,96 ],[ 67,100 ],
[ 68,82 ],[ 68,103 ],[ 69,89 ],[ 69,101 ],[ 70,83 ],[ 70,104 ],
[ 71,93 ],[ 71,101 ],[ 71,103 ],[ 72,96 ],[ 72,101 ],[ 73,89 ],
[ 73,91 ],[ 73,102 ],[ 74,93 ],[ 74,102 ],[ 75,82 ],[ 75,97 ],
[ 75,102 ],[ 76,83 ],[ 76,86 ],[ 76,98 ],[ 76,102 ],[ 77,100 ],
```

Fig. 24. Continued

```
[ 77,101 ],[ 78,91 ],[ 78,103 ],[ 79,83 ],[ 79,94 ],[ 79,103 ],
[ 80,89 ],[ 80,95 ],[ 80,103 ],[ 80,104 ],[ 81,105 ],[ 82,105 ],
[ 83,105 ],[ 84,105 ],[ 85,105 ],[ 86,105 ],[ 87,105 ],
[ 88,105 ],[ 89,105 ],[ 90,105 ],[ 91,105 ],[ 92,105 ],
[ 93,105 ],[ 94,105 ],[ 95,105 ],[ 96,105 ],[ 97,105 ],
[ 98,105 ],[ 99,105 ],[ 100,105 ],[ 101,105 ],[ 102,105 ],
[ 103,105 ],[ 104,105 ] ]
```

Fig. 24. Continued

This is the data which was used to draw Figure 19. The variable PL also lists the edges of the Hasse diagram but it also includes the edge label (recorded as a number indicating the position of the reflection occurs in the list R). Recall that inside each column in $\tilde{K}$ there is a bi-infinite spiral labeled by a bi-infinite sequence of edge labels. Because A acts cocompactly on $\tilde{K}$ each such sequence repeats and there are only finitely many patterns. From the data recorded in PL, it is straightforward to calculate this finite list of infinitely repeating patterns of edge labels which label the spirals winding their way through the columns (Figure 25).

```
gap> createColumns();

 There are 432 simplices
 There are 18 columns each containing 24 simplices
 There are 14 column patterns

Type paths; to see the reflections creating the simplices
Type columns; to see the sequences creating the columns
Type colPatterns; to see the list of column patterns

If this Dynkin diagram has symmetries, the column patterns can
be simplified by typing simplifyPatterns(s); where s is a
list of length 4 that the i-th generater should
be replaced with.  The list of patterns returned will be
cyclically minimal and without repetitions.
```

Fig. 25. GAP Run 6

In the type F_4 situation, the 18 distinct sequences of edge labels are as follows (Figure 26).

```
gap> columns;
[ [ 1,2,4,10,5,20,6,17,16,23,12,21,14,24,13,19,18,22,8,15,9,11,7,3],
  [ 1,2,6,4,5,20,12,6,16,23,13,12,14,24,8,13,18,22,7,8,9,11,4,7 ],
  [ 1,2,10,6,5,20,17,12,16,23,21,13,14,24,19,8,18,22,15,7,9,11,3,4 ],
  [ 1,3,2,4,5,10,20,6,16,17,23,12,14,21,24,13,18,19,22,8,9,15,11,7 ],
  [ 1,3,4,2,5,10,6,20,16,17,12,23,14,21,13,24,18,19,8,22,9,15,7,11 ],
  [ 1,4,2,10,5,6,20,17,16,12,23,21,14,13,24,19,18,8,22,15,9,7,11,3 ],
  [ 1,4,7,2,5,6,4,20,16,12,6,23,14,13,12,24,18,8,13,22,9,7,8,11 ],
  [ 1,4,10,16,5,6,17,14,16,12,21,18,14,13,19,9,18,8,15,1,9,7,3,5 ],
  [ 1,4,16,7,5,6,14,4,16,12,18,6,14,13,9,12,18,8,1,13,9,7,5,8 ],
  [ 1,6,4,16,5,12,6,14,16,13,12,18,14,8,13,9,18,7,8,1,9,4,7,5 ],
  [ 1,6,9,4,5,12,1,6,16,13,5,12,14,8,16,13,18,7,14,8,9,4,18,7 ],
  [ 1,6,13,9,5,12,8,1,16,13,7,5,14,8,4,16,18,7,6,14,9,4,12,18 ],
  [ 1,7,2,6,5,4,20,12,16,6,23,13,14,12,24,8,18,13,22,7,9,8,11,4 ],
  [ 1,7,3,2,5,4,10,20,16,6,17,23,14,12,21,24,18,13,19,22,9,8,15,11 ],
  [ 1,7,6,9,5,4,12,1,16,6,13,5,14,12,8,16,18,13,7,14,9,8,4,18 ],
  [ 1,7,9,3,5,4,1,10,16,6,5,17,14,12,16,21,18,13,14,19,9,8,18,15 ],
  [ 1,9,3,4,5,1,10,6,16,5,17,12,14,16,21,13,18,14,19,8,9,18,15,7 ],
  [ 1,13,7,9,5,8,4,1,16,7,6,5,14,4,12,16,18,6,13,14,9,12,8,18 ] ]
```

Fig. 26. GAP Run 7

Notice that each simplex in K uniquely determines a column (along the lines described in the previous chapter) which is why there are exactly 432 subsequences of length 4 in the 18 (repeating) sequences of length 24.

B. The reduction to the cross-section complex

In this section, I show that searching for nonpositively curved piecewise Euclidean metrics on K under which the vertical translation map becomes an isometry is equivalent to searching for nonpositively curved piecewise Euclidean metrics on the cross-section complex. In order to make this argument precise, I need to introduce some additional definitions and results from the theory of CAT(0) spaces.

Definition B.1 (Isometries). Let X be a metric space and let γ be an isometry on X. The *displacement function* measures how far γ moves each point of X. The *translation length* of γ is the infimum of its displacement. The *minset* of γ is the set

of points in X whose displacement is exactly the translation length. The isometry γ is called *elliptic* if γ fixes a point in X, it is *parabolic* if its translation length is 0 but γ is not elliptic, and it is *hyperbolic* if its translation length is positive.

This division mirrors the classification of isometries of hyperbolic space. It is a standard result that, if a group G acts on a CAT(0) space X cocompactly by isometries, then each element of G is either hyperbolic or elliptic [16]. In particular, if the group generated by A together with the vertical translation map t act on $\tilde{K}$ by isometries, then t will be a hyperbolic isometry. The following properties of minsets of hyperbolic isometries are proved in Chapter II.6 of [16].

Lemma B.2 (Properties of minsets). *If X is a* CAT(0) *metric space and γ is a hyperbolic isometry, then* $\textsc{Minset}(\gamma)$ *is a nonempty closed, complete, convex subset of X. In addition,* $\textsc{Minset}(\gamma)$ *is isomorphic to the direct product $Y \times \mathbf{R}$ and the restriction of γ to* $\textsc{Minset}(\gamma)$ *sends (y, t) to $(y, t + k)$ where k is the translation length of γ.*

Notice that, as a consequence of convexity, $\textsc{Minset}(\gamma)$ is also contractible. The following result shows that it is sufficient to study a piecewise Euclidean metric on the cross-section complex.

Theorem B.3 (Reduction to the cross-section). *Let A be an Artin group of finite-type and let K be its Brady-Krammer complex. There exists a piecewise Euclidean metric of nonpositive curvature on K under which the vertical translation map is an isometry if and only if there is a piecewise Euclidean metric of nonpositive curvature on the cross-section complex.*

Proof. The proof in the forward direction hinges on the fact that the dimension of the Brady Krammer complex K is identical to the cohomological dimension of the

group A (which is the number of vertices in the defining Dynkin diagram). The computation of the cohomological dimension is straightforward. The fact that K is a $K(A,1)$ shows the cohomological dimension is at most n. On the other hand, every n-generated finite-type Artin group contains a free abelian subgroup of rank n showing that the cohomological dimension is at least n. Suppose K has a piecewise Euclidean metric of nonpositive curvature under which the vertical translation map is an isometry (on $\tilde{K}$), and let M be the minset of the vertical translation map. By Lemma B.2 M is nonempty, convex, and contractible, and by symmetry it is invariant under the action of A on $\tilde{K}$. Moreover because A acts freely on $\tilde{K}$, it acts freely on M so that the quotient space M/A is a $K(A,1)$ sitting inside K. If $M/A \subset K$ does not contain the unique vertex of K, then there is a deformation retraction of M/A onto a lower dimensional complex, contradicting the cohomological dimension. Thus M/A contains the vertex of K; M contains all the vertices of $\tilde{K}$, and, because M is convex, it contains all of the simplices of $\tilde{K}$ as well. As a consequence $\tilde{K}$ is isometric to $Y \times \mathbf{R}$ and the vertical translation map only affects the second coordinate. The complex Y is, of course, the universal cover of the cross-section complex L. Since a product of geodesic metric spaces is CAT(0) if and only of each factor is CAT(0), the universal cover of the cross-section complex must support a piecewise Euclidean CAT(0) metric which descends to a nonpositively curved piecewise Euclidean metric on the cross-section complex itself.

The reverse direction is nearly immediate. A nonpositively curved piecewise Euclidean metric on the cross-section complex L lifts to a piecewise Euclidean CAT(0) metric on $\tilde{L}$ which in turn can be lifted column by column as follows. For each simplex σ in $\tilde{L}$ its preimage in $\tilde{K}$ is a column. Add a metric to this column so that it is isometric to $\sigma \times \mathbf{R}$. Moreover arrange the simplicial structure so that the difference in the second coordinates between adjacent vertices in the spiral is constant. If the

same constant is used for each column in $\tilde{K}$ then these metrics are compatible; $\tilde{K}$ is isometric to $\tilde{L}\times\mathbf{R}$, so is CAT(0), and the vertical translation map is an isometry. □

As an aside note that this result remains true and the proof is unchanged for the Charney-Meier-Whittlesey construction applied to a group with a Garside structure so long as the dimension of the constructed complex is equal to the cohomological dimension.

C. The tetrahedra and their dihedral angles

Theorem B.3 reduces the proof of Theorem A to a question about the existence of an appropriate metric on a finite 3-dimensional simplicial complex. Since the metric needs to be invariant under the automorphisms of L induced by the symmetry of the F_4 Dynkin diagram and by conjugation by w, we can reduce the number of metrically distinct columns we need to consider. In this section I show that the metrics on the cross-section complex we need to consider can be described by three numbers labeling the 24 edge lengths in four metric tetrahedra with only 13 possibly distinct dihedral angles because of the symmetries of the tetrahedra.

The first simplification is that, if one reflection r is the conjugate of another reflection s by a power of w, then there is a symmetry in $\tilde{K}$ which sends an edge labeled r to an edge labeled s, and in particular, the lengths of the projections of each of these edges in L must be equal. The conjugacy classes of R relative to w are recorded in the variable *wc*. In the type F_4 situation the 24 reflections fall into four classes (Figure 27).

```
gap> wc;
(1,5,16,14,18,9)(2,20,23,24,22,11)(3,10,17,21,19,15)(4,6,12,13,8,7)
```

Fig. 27. GAP Run 8

Thus when considering the edge labels defining columns we can replace the number indicating the actual reflection with a number indicating its conjugacy class relative to w. These simplified column descriptions are contained in the variable `colPatterns`. In type F_4 there are 14 such patterns (Figure 28).

```
gap> colPatterns;
[ [ 1,1,3,4,1,1,3,4,1,1,3,4,1,1,3,4,1,1,3,4,1,1,3,4 ],
  [ 1,1,4,3,1,1,4,3,1,1,4,3,1,1,4,3,1,1,4,3,1,1,4,3 ],
  [ 1,1,4,4,1,1,4,4,1,1,4,4,1,1,4,4,1,1,4,4,1,1,4,4 ],
  [ 1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4 ],
  [ 1,2,4,3,1,2,4,3,1,2,4,3,1,2,4,3,1,2,4,3,1,2,4,3 ],
  [ 1,2,4,4,1,2,4,4,1,2,4,4,1,2,4,4,1,2,4,4,1,2,4,4 ],
  [ 1,3,1,4,1,3,1,4,1,3,1,4,1,3,1,4,1,3,1,4,1,3,1,4 ],
  [ 1,3,2,4,1,3,2,4,1,3,2,4,1,3,2,4,1,3,2,4,1,3,2,4 ],
  [ 1,3,4,2,1,3,4,2,1,3,4,2,1,3,4,2,1,3,4,2,1,3,4,2 ],
  [ 1,4,1,4,1,4,1,4,1,4,1,4,1,4,1,4,1,4,1,4,1,4,1,4 ],
  [ 1,4,2,3,1,4,2,3,1,4,2,3,1,4,2,3,1,4,2,3,1,4,2,3 ],
  [ 1,4,2,4,1,4,2,4,1,4,2,4,1,4,2,4,1,4,2,4,1,4,2,4 ],
  [ 1,4,3,2,1,4,3,2,1,4,3,2,1,4,3,2,1,4,3,2,1,4,3,2 ],
  [ 1,4,4,2,1,4,4,2,1,4,4,2,1,4,4,2,1,4,4,2,1,4,4,2 ] ]
```

Fig. 28. GAP Run 9

A second simplification is derived from the symmetry of the F_4 Dynkin diagram. Under this symmetry the reflections in the first and the fourth classes switch places as do the reflections in the second and the third classes. In `coxeter.g` this simplification is acomplished using the command `simplifyPatterns` (Figure 29).

```
gap> simplifyPatterns([1,2,2,1]);
[ [ 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1 ],
  [ 1,1,1,2,1,1,1,2,1,1,1,2,1,1,1,2,1,1,1,2,1,1,1,2 ],
  [ 1,1,2,2,1,1,2,2,1,1,2,2,1,1,2,2,1,1,2,2,1,1,2,2 ],
  [ 1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2,1,2 ] ]
```

Fig. 29. GAP Run 10

One way to interprete this output is that each tetrahedron in L has a cycle of four edges which are projections of the edges in class 1 or class 2 according to the patterns listed above (i.e. (1111), (1112), (1122) or (1212)). For the sake of readability I call

the length of these edges in L a and b, respectively. See Figure 32. The other edges of the tetrahedra are projections of edges in K which are labeled by group elements which are the product of two reflections. The computer initially labels each diagonal with a different variable identifying only those which must be equal because of the structure of the labeled poset. In each case all of the remaining edges must have the same length which I call c. Thus there are only four types of metric tetrahedra, one for each of the four patterns listed above. In `coxeter.g` the variable `tetrahedra` stores the result of this computation (Figure 30). It lists the six edge lengths of each tetrahedra in the order shown in Figure 31.

```
gap> tetrahedra;
[ [ 1, 1, 1, 1, 5, 5 ], [ 1, 1, 1, 2, 5, 5 ], [ 1, 1, 2, 2, 5, 5 ],
[ 1, 2, 1, 2, 5, 5 ] ]
```

Fig. 30. GAP Run 11

Thus the numbers 1, 2 and 5 used by computer correspond to a, b and c in my diagrams. In addition to naming the lengths of the edges, we also need names for the dihedral angles. Figure 32 shows all four types of metric tetrahedra in the cross-section complex together with the edge names and the names of the dihedral angles.

Because of the symmetries in the tetrahedra, we have only 13 distinct dihedral angles. To make it easier to read I have given these thirteen dihedral angles names.

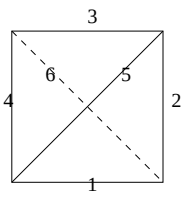


Fig. 31. Ordering the edges of a tetrahedron

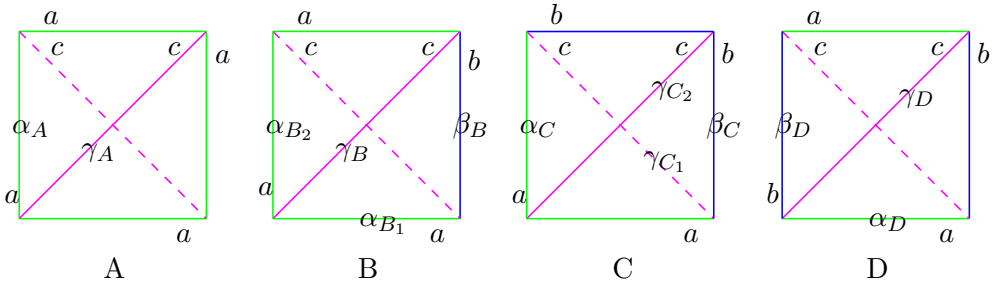


Fig. 32. The simplices in the cross-section for type F_4

The thirteen names, in order, are α_A, γ_A, α_{B_1}, α_{B_2}, β_B, γ_B, α_C, β_C, γ_{C_1}, γ_{C_2}, α_D, β_D and γ_D. The greek letter indicates the edge label, the subscript indicates the tetrahedron and an additional subscript is added if this does not uniquely determine the dihedral angle. For example, α_A is the dihedral angle between two sides joining at the left edge of the first tetrahedron. Because of the symmetries of the tetrahedra, A has two different dihedral angles, B has four, C has four and D has three different dihedral angles. In the program we use the command `dihAngForGp` to find all of the distinct dihedral angles for the complex (Figure 33).

```
gap> dihAngForGp();
The dihedral angles for the complex are
[ 1,1,1,1,2,2,3,4,3,5,6,6,7,7,8,8,9,10,11,12,11,12,13,13 ] .
```

Fig. 33. GAP Run 12

Each group of six numbers represents the six dihedral angles of a cross-section simplex. Dihedral angles which must be equal are given the same number. For example, we can see that the first simplex has only two different dihedral angles.

D. The edge links and their system of inequalities

In this section, I describe the finite graphs which are the links of edges in L and the system of linear inequalities involving their dihedral angles which would need to

be satisfied for a hypothetical piecewise Euclidean metric on L to be nonpositively curved. Because of all the symmetries of the noncrossing partition lattice of type F_4, only a small number of edge links need to be computed. Since L is a finite simplicial complex explicitly represented in the computer, as are the symmetries we wish to preserve, it is relatively straightforward to find a representative set of edges up to symmetry, and for each edge to calculate its link. In the program, the command `nodesCheck` generates a list of representative edges whose links need to be checked. Since there is a way to associate an element of the noncrossing partition lattice of type F_4 to each edge in L, the program uses the position of an element in V as the name of the edge. Recall that V is the list of 105 permutations below the Coxeter element (Figure 34).

```
gap> nodesCheck();
[ 18, 11, 28, 33, 26, 39, 43 ]
```

Fig. 34. GAP Run 13

Each arc in the finite graph which is the link of an edge can be labeled by the dihedral angle which measures its length. In the case of F_4, there are 7 edge link types, as we have seen, and their links are shown in Figures 35 through 41.

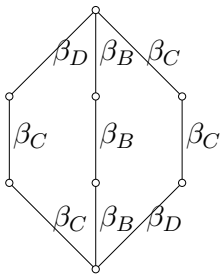


Fig. 35. $lk(11)$ in F_4

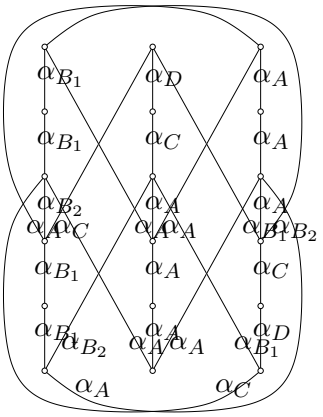


Fig. 36. $lk(18)$ in F_4

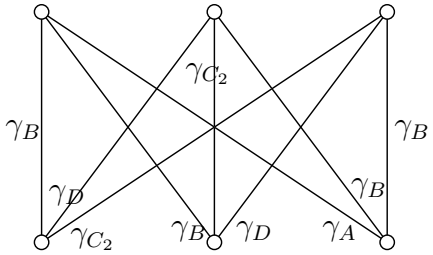


Fig. 37. $lk(26)$ in F_4

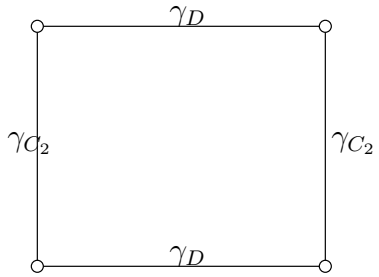


Fig. 38. $lk(28)$ in F_4

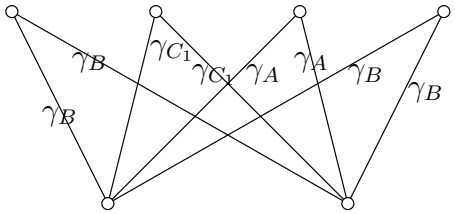


Fig. 39. $lk(33)$ in F_4

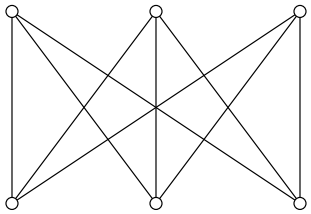


Fig. 40. $lk(39)$ in F_4 with all edges labeled by γ_A

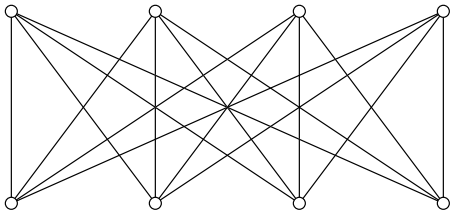


Fig. 41. $lk(43)$ in F_4 with all edges labeled by β_D

We have 7 edge links to check for closed geodesic loops of length less than 2π. Notice that the shortest closed geodesic loop in a finite metric graph is always a simple closed loop. Thus, in order to check Gromov's link condition, it is sufficient to check whether every simple closed loop has length at least 2π. Once a graph have been found, the program finds all simple closed loops in the graph using the command `findAllSimpleClosedLoopsForNode` (Figure 42).

```
gap> findAllSimpleClosedLoopsForNode(11);
[ [ 30, 81, 31, 93, 58, 92, 30 ], [ 30, 81, 32, 90, 58, 92, 30 ],
  [ 30, 92, 58, 90, 32, 81, 30 ], [ 30, 92, 58, 93, 31, 81, 30 ],
  [ 31, 81, 32, 90, 58, 93, 31 ], [ 31, 93, 58, 90, 32, 81, 31 ] ]
```

Fig. 42. GAP Run 14

The three simple closed loops in $lk(11)$, for example, are double counted once with each orientation. For each simple closed loop the program records a linear inequality which asserts that the length of this loop is at least 2π. This system is redundant in the sense that many of the inequalities are implied by other inequalities in the system. The command `simplifiedReducedIneqes` finds the full list and then removes those which are obviously redundant. In the case of F_4, the program produces 27 inequalities as follows (Figure 43).

```
[ 1: [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 2 ],
  2: [ 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 2, 0 ],
  3: [ 0, 0, 0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0 ],
  4: [ 0, 0, 0, 0, 0, 2, 0, 0, 0, 1, 0, 0, 1 ],
  5: [ 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0 ],
  6: [ 0, 0, 0, 0, 3, 0, 0, 2, 0, 0, 0, 1, 0 ],
  7: [ 0, 0, 1, 1, 0, 0, 3, 0, 0, 0, 1, 0, 0 ],
  8: [ 0, 0, 2, 1, 0, 0, 2, 0, 0, 0, 1, 0, 0 ],
  9: [ 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0 ],
 10:[ 0, 0, 3, 2, 0, 0, 1, 0, 0, 0, 0, 0, 0 ],
 11:[ 0, 1, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 1 ],
 12:[ 0, 1, 0, 0, 0, 2, 0, 0, 0, 1, 0, 0, 0 ],
 13:[ 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
```

Fig. 43. GAP Run 15

```
14:[ 0, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0 ],
15:[ 0, 2, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0 ],
16:[ 3, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0 ],
17:[ 3, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
18:[ 2, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0 ],
19:[ 2, 0, 2, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0 ],
20:[ 2, 0, 4, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
21:[ 2, 0, 3, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 ],
22:[ 3, 0, 0, 0, 0, 0, 2, 0, 0, 0, 1, 0, 0 ],
23:[ 4, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 ],
24:[ 4, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
25:[ 5, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
26:[ 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ],
27:[ 4, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0 ] ].
```

Fig. 43. Continued

The numbers listed are the coefficients of the 13 dihedral angles (in order) which occur on the left hand side of the inequality. The left hand side is always at least as big as the right hand side which is equal to 2π. For example, inequality (16) asserts that $3\alpha_A + 1\alpha_{B_1} + 1\alpha_{B_2} + 1\alpha_C \geq 2\pi$. As a final comment, notice that all of the inequalities are valid when $a = b$ and $2a = \sqrt{3}c$ since this causes all of dihedral angles to be either $\frac{\pi}{3}$ (if it is an α or β) or $\frac{\pi}{2}$ (if it is a γ).

E. Solving the system and completing the proof

The linear system of inequalities for F_4 produced by the program is small enough to analyze by hand. In particular, using the dihedral angle comparison lemma, I show that the only piecewise Euclidean metric on the cross-section which respects the symmetry of the F_4 Dynkin diagram and has no closed geodesic loops of length less than 2π in the edge links is the one derived from the natural metric. Once this has been shown, the remainder of the proof is brief since the natural metric has previously been analyzed. The dihedral angle comparison lemma alluded to above is the following.

Lemma E.1 (Dihedral angle comparisons). *If σ and τ are two Euclidean n-dimensional simplices whose vertices have been identified so that each dihedral angle in σ is at least as big as the corresponding dihedral angle in τ, then σ and τ are similar.*

Proof. Let $\mathbf{u}_1, \ldots, \mathbf{u}_n$ be the outward pointing unit normal vectors for σ and let $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be the corresponding outward pointing unit normal vectors for τ. The dihedral angle between the $\mathbf{u}_i$ face and the $\mathbf{u}_j$ face is $\pi - \arccos(\mathbf{u}_i \cdot \mathbf{u}_j)$, so the angle comparisons are encoded in the inequalities $\mathbf{u}_i \cdot \mathbf{u}_j \geq \mathbf{v}_i \cdot \mathbf{v}_j$ for all i and j.

Next, the fact that $\{\mathbf{u}_i\}$ come from outward normals of a simplex implies, by Minkowski's theorem, that there exist positive a_i such that $\sum a_i \mathbf{u}_i = \mathbf{0}$. Up to a scale factor, the value a_i is the volume of the $\mathbf{u}_i$ face and the fact that the sum equals zero is a consequence of the divergence theorem. Consider the following chain of inequalities:

$$\begin{aligned} 0 &= \left\| \sum a_i \mathbf{u}_i \right\|^2 = \sum a_i a_j \mathbf{u}_i \cdot \mathbf{u}_j \geq \sum a_i a_j \mathbf{v}_i \cdot \mathbf{v}_j \\ &= \left\| \sum a_i \mathbf{v}_i \right\|^2 \geq 0. \end{aligned}$$

If any of the inequalities $\mathbf{u}_i \cdot \mathbf{u}_j \geq \mathbf{v}_i \cdot \mathbf{v}_j$ were strict, this would be a contradiction. Therefore, $\mathbf{u}_i \cdot \mathbf{u}_j = \mathbf{v}_i \cdot \mathbf{v}_j$. As a result, all of the corresponding diheral angles are equal and σ and τ are similar. □

Throughout the remainder of this section, assume that K has been given a piecewise Euclidean metric that respects the symmetry induced by the symmetry of F_4 Dynkin diagram and assume that none of the edge links of L contain any closed geodesic loops of length less than 2π. Under these assumptions, it is straightforward to see that tetrahedron A must be similar to a tetrahedron with the $\tilde{A}_3$ Coxeter metric as follows.

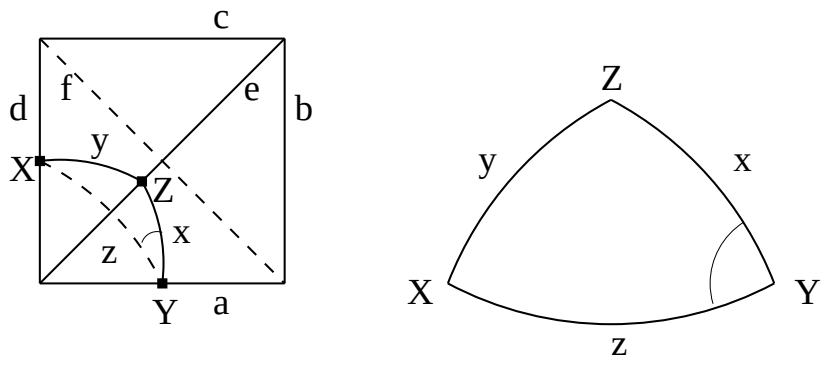


Fig. 44. A Euclidean tetrahedron and a spherical triangle

Lemma E.2. *Tetrahedron A is similar to a tetrahedron with the $\tilde{A}_3$ Coxeter metric.*

Proof. By inequalities (13) and (26), $\gamma_A \geq \frac{\pi}{2}$ and $\alpha_A \geq \frac{\pi}{3}$, respectively. This shows that the dihedral angles in tetrahedron A are at least as large as those in the Coxeter shape. Lemma E.1 completes the proof. □

As a consequence the edge lengths a and c satisfy the equation $4a^2 = 3c^2$. If we can show that a must equal b, than all four tetrahedra will have this Coxeter metric. I do this using a little bit of spherical trigonometry. Let T be the Euclidean tetrahedron shown in Figure 44 and let the labels a through f denote the length of each edge. In addition, let X, Y and Z be the vertices of the spherical triangle which is the vertex link shown with edge lengths x, y and z. By the Euclidean law of cosines, $a^2 + e^2 - b^2 = 2ae\cos x$, for example. Similarly, the spherical law of cosines implies that $\cos z - \cos x \cos y = \sin x \sin y \cos Z$.

Remark E.3. If we identify the tetrahedron B (see Figure 32) with the tetrahedron shown in Figure 44, i.e. replacing labels c and d with a and replacing labels e and f with c, where a and c satisfy the equation $2a = \sqrt{3}c$, then we can already calculate the sine and cosine of y and z since the shapes of these triangles are known. In particular,

it is routine to check that $\cos y = \frac{1}{\sqrt{3}}$, $\sin y = \frac{\sqrt{2}}{\sqrt{3}}$, $\cos z = \frac{1}{3}$, and $\sin z = \frac{2\sqrt{2}}{3}$.

Lemma E.4. *Under our assumptions on the metric, we must have $a = b$.*

Proof. From inequality (5), we have that $\gamma_B \geq \frac{\pi}{2}$ and thus $\cos \gamma_B \leq 0$. Since γ_B corresponds to the angle at Z in the spherical triangle shown, by the spherical law of cosines,

$$\cos \gamma_B = \cos Z = \frac{\cos z - \cos x \cos y}{\sin x \sin y} \leq 0.$$

Using the values listed in Remark E.3, and solving for $\cos x$, we find that $\cos x \geq \frac{1}{\sqrt{3}}$. Next, consider inequality (27). Since tetrahedron A has the $\tilde{A}_3$ metric, $\alpha_A = \frac{\pi}{3}$, inequality (27) implies $\alpha_{B_2} \geq \frac{\pi}{3}$, and thus $\cos \alpha_{B_2} \leq \frac{1}{2}$. By the spherical law of cosine,

$$\cos \alpha_{B_2} = \cos X = \frac{\cos x - \cos y \cos z}{\sin y \sin z} \leq \frac{1}{2}.$$

Using the values listed in Remark E.3, and solving for $\cos x$, we find that $\cos x \leq \frac{1}{\sqrt{3}}$. Thus $\cos x = \frac{1}{\sqrt{3}}$ which happens to be $\cos y$. Since $2ac \cos x = a^2 + c^2 - b^2$ and $2ac \cos y = c^2$, and a and b are both positive, we have that $\cos x = \cos y$ is true if and only if $a = b$. □

To conclude, I have shown that the only piecewise Euclidean metric on the cross-section complex L which respects its symmetries and which contains no closed geodesic loops of length less than 2π in the finite graphs which are the edge links of L is the one induced by the natural metric on K. The link of the vertex in L in this metric has already been analyzed by Tom Brady and Jon McCammond and found to contain a closed geodesic loop of length less than 2π. An implementation of their method to test complexes built using the natural metric is contained in `coxeter.g` under the name `testGroup`. In the F_4 case, the results are as follows (Figure 45).

```
gap> testGroup();

  The type (F,4) Brady-Krammer complex is not CAT(0)
  using the standard metric because some triples of reflections
  give rise to short geodesic cycles in the unique vertex link

  There are 12 bad triples of reflections
  There are 2 bad patterns of reflections

  To see the bad triples type badTri;
  To see the bad patterns type badPatterns;
```

Fig. 45. GAP Run 16

Since the only piecewise Euclidean symmetry-respecting metric which satisfies Gromov's link condition on the edge links does not satisfy the link condition on the vertex link, the cross-section complex does not support a piecewise Euclidean symmetry-respecting metric. By Theorem B.3, this completes the proof of Theorem A.

CHAPTER V

THE BRADY-KRAMMER COMPLEX OF TYPE D_4

The main goal of this chapter is to prove the following theorem.

Theorem B (Type D_4). *The Brady-Krammer complex for the Artin group of type D_4 does not support a piecewise Euclidean metric of nonpositive curvature which respects all of the symmetries of the complex.*

The argument is parallel to, but slightly different from, the argument in the previous chapter. Thoughout this chapter let A denote the Artin group of type D_4, W the Coxeter group of type D_4, and K the Brady-Krammer complex of type D_4. The group W has 192 elements and acts as a finite reflection group on $\mathbf{R}^4$ with 12 reflections. The lattice of noncrossing partition of type D_4 has 50 elements, arranged in 5 levels in the pattern $1 - 12 - 24 - 12 - 1$. The Dynkin diagram of type D_4 is shown in Fig. 46.

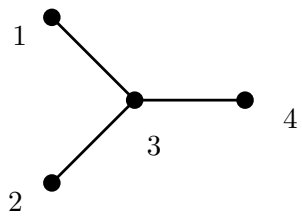


Fig. 46. The Dynkin diagram of type D_4

Even though the Coxeter group of type D_4 is less than one-fifth the size of the Coxeter group of type F_4 and even though its Dynkin diagram has more symmetries (and the noncrossing partition lattice of type D_4 shown in Figure 47 is noticeably simpler than the noncrossing partition lattice of type F_4), the analysis in this case

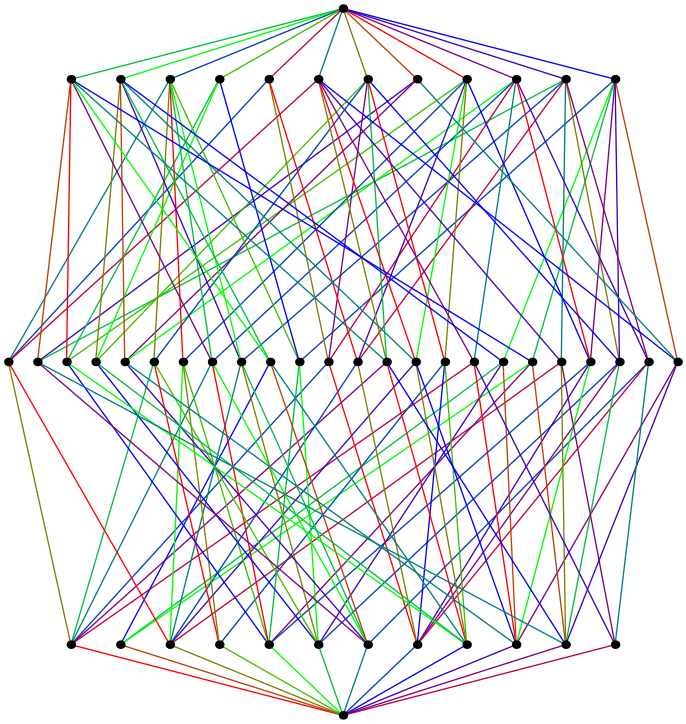


Fig. 47. The lattice of noncrossing partitions of type D_4

is slightly more complicated. The structure of the chapter is parallel to that of the previous chapter but shorter since the notations and conventions have already been defined and general results, such as the reduction to the cross-section complex and the dihedral angle comparison lemma do not need to be reproved. In Section A, I describe the structure of K. In section B, I describe the tetrahedra in the cross-section complex and the possible metrics which would respect its symmetries. In Section C, the links of the edges in the cross-section are examined, and a finite system of linear inequalities is produced which encodes the restrictions on the dihedral angles of the tetrahedra used to construct the cross-section if the metric space under consideration is to be nonpositively curved. In the final section, I analize this system of inequalities and show that none of its solutions lead to a nonpositively curved metric on the

cross-section.

A. The structure of the D_4 Poset

The basic information about the noncrossing partition lattice of type D_4 is the following (Figure 48).

```
gap> createCoxeterInfo("D",4);

type = Coxeter group ("D",4)
R    = a list of the 12 reflections of W
w    = coxeter element ([ 1, 2, 4, 3 ])

c  = conjugacy table for R
wc = a list of the conjugates classes for R rel w

L  = a graded list of permutations below w ([ 1, 12, 24, 12, 1 ])
V  = a list of the 50 permutations listed in L

P  = poset structure for L
PL = poset structure for L with labels

Other information is stored in len, clen, and cw --but
these are mostly for internal use.

  To calculate the column structure, type createColumns();

  Since this group is rank 4, try testGroup(); to see whether
  the Brady-Krammer complex for the Artin group of this type
  is CAT(0) using the standard coxeter shape metric.
```

Fig. 48. GAP Run 17

As noted in the last chapter, the way GAP represents finite Coxeter groups internally is as a permutation group permuting the root system. In the case of D_4, there are 12 reflections and thus 24 roots. As before R records the list of reflections and w records the permutation which represents the action of a Coxeter element on the 24 roots. The Coxeter element chosen is the standard one which multiplies the first, second, fourth and third reflections in the list of 12 reflections in this order. These first

four reflections correspond to the four vertices of the Dynkin diagram (Figure 49).

```
gap> w;
(1,17,22,13,5,10)(2,18,21,14,6,9)(3,12,11,15,24,23)(4,19,20,16,7,8)
```

Fig. 49. GAP Run 18

From the permutation, it is clear the order of w is 6. The 50 permutations which occur as a product of an initial segment of one of these sequences are listed in V. Thus $V[1]$ is the identity permutation, the elements $V[2]$ through $V[13]$ are the reflections, $V[14]$ through $V[37]$ are the permutations below w which are a product of two reflections, $V[38]$ through $V[49]$ are the permutations below w which are a product of three reflections and $V[50]$ is the permutation w itself. The edges of the Hasse diagram used to draw Figure 47 are listed in P (Figure 50).

The finite list of columns can be calculated as before (Figure 51).

The 18 types of columns in this case are as follows (Figure 52).

```
gap> columns;
```

B. The tetrahedra and their dihedral angles

Theorem B.3 once again reduces the proof of Theorem B to a question about the existence of an appropriate metric on a finite 3-dimensional simplicial complex. Since the metric needs to be invariant under the automorphisms of L induced by the symmetries of the D_4 Dynkin diagram and by conjugation by w, we can reduce the number of metrically distinct columns we need to consider. In this section I show that the metrics on the cross-section complex we need to consider can be described by four numbers labeling the 18 edge lengths in three metric tetrahedra with only 9 possibly distinct dihedral angles because of the symmetries of the tetrahedra.

The first simplification is that if one reflection r is the conjugate of another

```
gap> P;
[ [ 1, 2 ],[ 1,3 ],[ 1,4 ],[ 1,5 ],[ 1,6 ],[ 1,7 ],[ 1,8 ],
  [ 1,9 ],[ 1,10 ],[ 1,11 ],[ 1,12 ],[ 1,13 ],[ 2,14 ],[ 2,19 ],
  [ 2,21 ],[ 2,25 ],[ 2,27 ],[ 2,30 ],[ 3,23 ],[ 3,31 ],
  [ 3,32 ],[ 4,14 ],[ 4,20 ],[ 4,22 ],[ 4,26 ],[ 4,28 ],
  [ 4,33 ],[ 5,19 ],[ 5,20 ],[ 5,29 ],[ 6,16 ],[ 6,20 ],
  [ 6,21 ],[ 6,24 ],[ 6,32 ],[ 6,34 ],[ 7,17 ],[ 7,19 ],
  [ 7,22 ],[ 7,24 ],[ 7,31 ],[ 7,35 ],[ 8,15 ],[ 8,18 ],
  [ 8,21 ],[ 8,22 ],[ 8,23 ],[ 8,36 ],[ 9,25 ],[ 9,26 ],
  [ 9,29 ],[ 9,34 ],[ 9,35 ],[ 9,36 ],[ 10,16 ],[ 10,17 ],
  [ 10,23 ],[ 10,27 ],[ 10,28 ],[ 10,29 ],[ 11,18 ],[ 11,28 ],
  [ 11,30 ],[ 11,31 ],[ 11,34 ],[ 11,37 ],[ 12,15 ],[ 12,27 ],
  [ 12,32 ],[ 12,33 ],[ 12,35 ],[ 12,37 ],[ 13,30 ],[ 13,33 ],
  [ 13,36 ],[ 14,40 ],[ 14,42 ],[ 14,43 ],[ 15,38 ],[ 15,45 ],
  [ 15,48 ],[ 16,38 ],[ 16,41 ],[ 16,46 ],[ 17,39 ],[ 17,41 ],
  [ 17,44 ],[ 18,39 ],[ 18,45 ],[ 18,47 ],[ 19,40 ],[ 19,44 ],
  [ 20,40 ],[ 20,46 ],[ 21,38 ],[ 21,40 ],[ 21,47 ],[ 22,39 ],
  [ 22,40 ],[ 22,48 ],[ 23,38 ],[ 23,39 ],[ 24,40 ],[ 24,41 ],
  [ 24,49 ],[ 25,42 ],[ 25,44 ],[ 25,47 ],[ 26,42 ],[ 26,46 ],
  [ 26,48 ],[ 27,38 ],[ 27,43 ],[ 27,44 ],[ 28,39 ],[ 28,43 ],
  [ 28,46 ],[ 29,44 ],[ 29,46 ],[ 30,43 ],[ 30,47 ],[ 31,39 ],
  [ 31,49 ],[ 32,38 ],[ 32,49 ],[ 33,43 ],[ 33,48 ],[ 34,46 ],
  [ 34,47 ],[ 34,49 ],[ 35,44 ],[ 35,48 ],[ 35,49 ],[ 36,47 ],
  [ 36,48 ],[ 37,43 ],[ 37,45 ],[ 37,49 ],[ 38,50 ],[ 39,50 ],
  [ 40,50 ],[ 41,50 ],[ 42,50 ],[ 43,50 ],[ 44,50 ],[ 45,50 ],
  [ 46,50 ],[ 47,50 ],[ 48,50 ],[ 49,50 ] ]
```

Fig. 50. GAP Run 19

reflection s by a power of w then there is a symmetry in $\tilde{K}$ which sends an edge labeled r to an edge labeled s, and in particular, the lengths of the projections of each of these edges in L must be equal. The conjugacy classes of R relative to w are recorded in the variable wc. In the type D_4 situation the 12 reflections fall into four classes (Figure 53).

Thus when considering the edge labels defining columns we can replace the number indicating the actual reflection with a number indicating its conjugacy class relative to w. These simplified column descriptions are contained in the variable `colPatterns`. In type D_4 there are 15 such patterns (Figure 54).

```
gap> createColumns();

 There are 162 simplices
 There are 3 columns each containing 6 simplices
 There are 12 columns each containing 12 simplices
 There are 15 column patterns
Type paths; to see the reflections creating the simplices
Type columns; to see the sequences creating the columns
Type colPatterns; to see the list of column patterns

If this Dynkin diagram has symmetries, the column patterns can
be simplified by typing simplifyPatterns(s); where s is a
list of length 4 that the i-th generater should
be replaced with.  The list of patterns returned will be
cyclically minimal and without repetitions.
```

Fig. 51. GAP Run 20

```
[ [ 1, 2, 3, 7, 5, 6, 12, 8, 10, 9, 11, 4 ],
  [ 1, 2, 4, 3, 5, 6, 7, 12, 10, 9, 8, 11 ],
  [ 1, 2, 7, 4, 5, 6, 8, 7, 10, 9, 4, 8 ],
  [ 1, 3, 6, 7, 5, 12, 9, 8, 10, 11, 2, 4 ],
  [ 1, 3, 7, 6, 5, 12, 8, 9, 10, 11, 4, 2 ],
  [ 1, 4, 2, 3, 5, 7, 6, 12, 10, 8, 9, 11 ],
  [ 1, 4, 3, 6, 5, 7, 12, 9, 10, 8, 11, 2 ],
  [ 1, 4, 6, 2, 5, 7, 9, 6, 10, 8, 2, 9 ],
  [ 1, 6, 2, 7, 5, 9, 6, 8, 10, 2, 9, 4 ],
  [ 1, 6, 7, 10, 5, 9, 8, 1, 10, 2, 4, 5 ],
  [ 1, 6, 10, 2, 5, 9 ],
  [ 1, 7, 4, 6, 5, 8, 7, 9, 10, 4, 8, 2 ],
  [ 1, 7, 6, 10, 5, 8, 9, 1, 10, 4, 2, 5 ],
  [ 1, 7, 10, 4, 5, 8 ],
  [ 2, 7, 9, 4, 6, 8 ] ]
```

Fig. 52. GAP Run 21

```
gap> wc;
(1,5,10)(2,6,9)(3,12,11)(4,7,8)
```

Fig. 53. GAP Run 22

```
gap> colPatterns;
[ [ 1, 1, 2, 4, 1, 1, 2, 4, 1, 1, 2, 4 ],
  [ 1, 1, 4, 2, 1, 1, 4, 2, 1, 1, 4, 2 ],
  [ 1, 2, 1, 2, 1, 2 ],
  [ 1, 2, 2, 4, 1, 2, 2, 4, 1, 2, 2, 4 ],
  [ 1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4 ],
  [ 1, 2, 4, 3, 1, 2, 4, 3, 1, 2, 4, 3 ],
  [ 1, 2, 4, 4, 1, 2, 4, 4, 1, 2, 4, 4 ],
  [ 1, 3, 2, 4, 1, 3, 2, 4, 1, 3, 2, 4 ],
  [ 1, 3, 4, 2, 1, 3, 4, 2, 1, 3, 4, 2 ],
  [ 1, 4, 1, 4, 1, 4 ],
  [ 1, 4, 2, 2, 1, 4, 2, 2, 1, 4, 2, 2 ],
  [ 1, 4, 2, 3, 1, 4, 2, 3, 1, 4, 2, 3 ],
  [ 1, 4, 3, 2, 1, 4, 3, 2, 1, 4, 3, 2 ],
  [ 1, 4, 4, 2, 1, 4, 4, 2, 1, 4, 4, 2 ],
  [ 2, 4, 2, 4, 2, 4 ] ]
```

Fig. 54. GAP Run 23

A second simplification is derived from the symmetries of the D_4 Dynkin diagram. Under these symmetries the reflections in the first, the second and the fourth classes can be permuted at will, and the third class is always sent to itself. This simplification is acomplished using the command `simplifyPatterns` (Figure 55).

```
gap> simplifyPatterns([1,1,2,1]);
[ [ 1, 1, 1, 1, 1, 1 ],
  [ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 ],
  [ 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 2 ] ]
```

Fig. 55. GAP Run 24

One way to interprete this output is that each tetrahedron in L has a cycle of four edges which are projections of the edges in class 1 or class 2 according to the patterns listed above (i.e. (1111) or (1112)). For the sake of readability I call the length of these edges in L a and b, respectively. See Figure 57. The other edges of the tetrahedra are projections of edges in K which are labeled by group elements which are the product of two reflections. The computer initially labels each diagonal with a different variable identifying only those which must be equal because of the structure

of the labeled poset. This time, the remaining edges separate into two distinct classes whose lengths I call c and d. Although it need not have been the case, in the end there are only three types of metric tetrahedra, one for each of the three patterns listed above. The variable `tetrahedra` stores the result of this computation (Figure 56).

```
gap> tetrahedra;
[ [ 1, 1, 1, 1, 8, 8 ], [ 1, 1, 1, 1, 5, 8 ], [ 1, 1, 1, 2, 5, 5 ] ]
```

Fig. 56. GAP Run 25

The numbers 1, 2, 5 and 8 used by computer correspond to a, b, c and d in my diagrams. In addition to naming the lengths of the edges, we also need names for the dihedral angles. Figure 57 shows all three types of metric tetrahedra in the cross-section complex together with the edge names and the names of the dihedral angles.

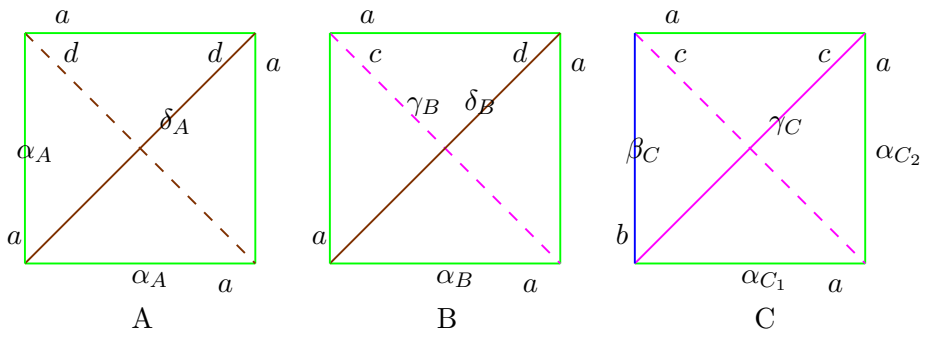


Fig. 57. The simplices in the cross-section for type D_4

Because of the symmetries in the tetrahedra, we have only 9 distinct dihedral angles. To make it easier to read I have given these nine dihedral angles names following the same conventions as before. The names of nine dihedral angles, in order, are α_A, δ_A, α_B, γ_B, δ_B, α_{C_1}, α_{C_2}, β_C and γ_C. Recall that the command `dihAngForGp` is used to find all of the distinct dihedral angles for the complex (Figure 58).

```
gap> dihAngForGp();
The dihedral angles for the complex are
[ 1,1,1,1,2,2,3,3,3,3,4,5,6,7,6,8,9,9 ] .
```

Fig. 58. GAP Run 26

C. The edge links and their system of inequalities

In this section, I describe the finite graphs which are the links of edges in L and the system of linear inequalities involving their dihedral angles which need to be satisfied for the metric on L to be nonpositively curved. The graphs and the system of inequalities are found as before. Using the command `nodesCheck` we find that there are four types of edge links (Figure 59).

```
gap> nodesCheck();
[ 9, 5, 14, 21 ]
```

Fig. 59. GAP Run 27

The four edge links are shown in Figures 60 throuth 63.

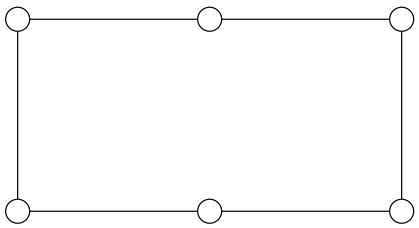


Fig. 60. $lk(5)$ in D_4 with all edges labeled by β_C

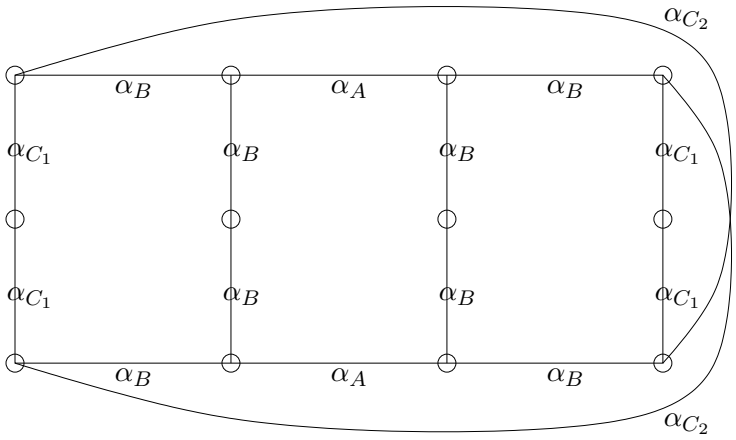


Fig. 61. $lk(9)$ in D_4

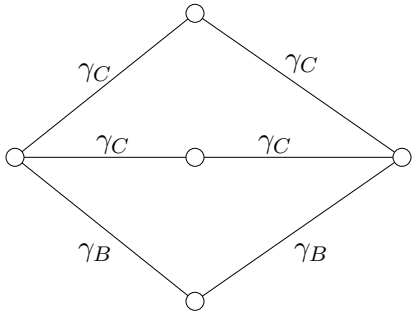


Fig. 62. $lk(14)$ in D_4

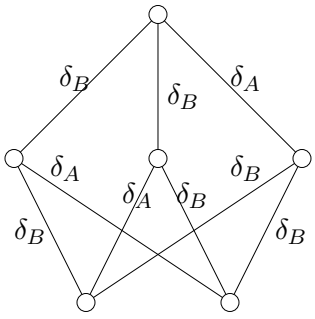


Fig. 63. $lk(21)$ in D_4

Using the command `simplifiedReducedIneqes` produces a system of 11 inequalities (Figure 64).

```
[ 1: [ 0, 0, 0, 0, 0, 0, 0, 0, 4 ],
  2: [ 0, 0, 0, 0, 0, 0, 0, 6, 0 ],
  3: [ 0, 0, 0, 0, 0, 4, 2, 0, 0 ],
  4: [ 0, 0, 0, 0, 6, 0, 0, 0, 0 ],
  5: [ 0, 0, 0, 2, 0, 0, 0, 0, 2 ],
  6: [ 0, 0, 4, 0, 0, 2, 0, 0, 0 ],
  7: [ 0, 1, 0, 0, 3, 0, 0, 0, 0 ],
  8: [ 0, 2, 0, 0, 2, 0, 0, 0, 0 ],
  9: [ 1, 0, 2, 0, 0, 2, 1, 0, 0 ],
 10: [ 1, 0, 4, 0, 0, 0, 1, 0, 0 ],
 11: [ 2, 0, 4, 0, 0, 0, 0, 0, 0 ] ]
```

Fig. 64. GAP Run 28

D. Solving the system and completing the proof

The linear system of inequalities is once again reasonably small, but this system is harder to analyze without the aid of computer. One source of this difficulty is that the inequalities almost always involve more than one tetrahedron making it difficult to apply the dihedral angle comparison lemma. As a substitute I have written Matlab code to help with the analysis. The general outline is similar to the analysis in the previous chapter in that I eventually prove that the only piecewise Euclidean symmetry-respecting metric on the cross-section complex whose edge links do not contain any closed geodesic loops of length less than 2π is the natural metric. The proof consists of three steps.

Lemma D.1 (Eliminating b). *Under our assumptions on the metric, we must have* $b^2 = 3(c^2 - a^2)$ *and as a consequence* $c > a$, $\gamma_C = \frac{\pi}{2}$ *and* $\beta_C = \frac{\pi}{3}$.

Proof. From inequality (1), $\gamma_C \geq \frac{\pi}{2}$ so that

$$0 \geq \cos\gamma_C = \frac{\cos z - \cos x \cos y}{\sin x \sin y},$$

where x, y and z are the appropriate face angles. This is equivalent to the inequality $\cos x \cos y \geq \cos z$. Rewriting this in terms of a, b and c, we find that $b^2 \leq 3(c^2 - a^2)$. Similarly, from inequality (2), $\beta_C \geq \frac{\pi}{3}$ and thus

$$\frac{1}{2} \geq \cos\beta_C = \frac{\cos z - \cos x \cos y}{\sin x \sin y},$$

where x, y and z are the appropriate face angles. This is equivalent to the inequality $\sin x \sin y + 2\cos x \cos y \geq 2\cos z$. Since x and y happen to lie in isometric triangles, $\cos x \cos y - \sin x \sin y = \cos(x+y)$ is equal to $\cos(\pi - u) = -\cos u$ where u is the third angle in this triangle. Thus an equivalent inequality is $3\cos x \cos y + \cos u \geq 2\cos z$. Rewriting all of these cosines in terms of a, b and c, clearing the denominators and simplifying the result, we find that this inequality is true if and only if

$$(b^2 - 3(c^2 - a^2))(b^2 + c^2 - a^2) \geq 0.$$

Since $3(c^2 - a^2) \geq b^2 > 0$, we know that the first factor is nonpositive and that $c^2 > a^2$. As a consequence $b^2 + c^2 - a^2 > 0$ and the second factor is positive. The only way this can happen is if the first factor is 0. This reduces a variable and also gives that $\gamma_C = \frac{\pi}{2}$ and $\beta_C = \frac{\pi}{3}$ as consequences. □

Lemma D.2 (Eliminating c). *Under our assumptions on the metric, we must have* $4a^2 = c^2 + 2d^2$.

Proof. Since $\gamma_C = \frac{\pi}{2}$, inequality (5) can be written as $\gamma_B \geq \frac{\pi}{2}$, $\cos\gamma_B \leq 0$, and with a simple calculation, this implies that $4a^2 \leq c^2 + 2d^2$. If this inequality is strict, then the Matlab code in `ineqTest.m` shows that inequality (6) fails. Thus $4a^2 = c^2 + 2d^2$. □

Lemma D.3 (Coxeter shapes). *Under our assumptions on the metric, all tetrahedra are similar to Coxeter shapes of type* $\tilde{A}_3$.

Proof. By using `ineqTest.m`, $2a > \sqrt{3}d$ implies that inequality (6) fails and $2a < \sqrt{3}d$ causes inequalities (7) and (10) to fail. Thus $2a = \sqrt{3}d$ and we get $c = d$, $a = b$. This makes all three simplices similar to the Coxeter shape. □

To conclude, I have shown that the only piecewise Euclidean metric on the cross-section complex L which respects its symmetries and which contains no closed geodesic loops of length less than 2π in the finite graphs which are the edge links of L is the one induced by the natural metric on K. The link of the vertex in L in this metric has already been analyzed by Tom Brady and Jon McCammond and found to contain a closed geodesic loop of length less than 2π. An implementation of their method to test complexes built using the natural metric is contained in `coxeter.g` under the name `testGroup`. In the D_4 case, the results are as follows (Figure 65).

```
gap> testGroup();

  The type (D,4) Brady-Krammer complex is not CAT(0)
  using the standard metric because some triples of reflections
  give rise to short geodesic cycles in the unique vertex link

  There are 3 bad triples of reflections
  There are 3 bad patterns of reflections

  To see the bad triples type badTri;
  To see the bad patterns type badPatterns;
```

Fig. 65. GAP Run 29

Since the only piecewise Euclidean symmetry-respecting metric which satisfies Gromov's link condition on the edge links does not satisfy the link condition on the vertex link, the cross-section complex does not support a piecewise Euclidean symmetry-respecting metric. By Theorem B.3, this completes the proof of Theo-

rem B.

Finally, Theorem C is an immediate corollary of Theorems A and B. As mentioned in the introduction Ruth Charney, John Meier and Kim Whittlesey have shown in [21] that the cross-section complex of each Brady-Krammer complex admits an asymmetric metric satisfying a weak version of nonpositive curvature using a technique pioneered by Mladen Bestvina in [5]. The "distances" they define are on ordered pairs of the vertices of the cross-section complex rather than on ordered pairs of arbitrary points. Their distance fuction is nonnegative, 0 only on pairs of identical points, and satisfies the triangle inequality. Nevertheless it is not a metric since the "distance" from x to y is often slightly different from the "distance" from y to x. Despite this defect of the metric, the CAT(0)-like condition they use enables them to establish many of the structural results about the groups which would follow naturally from a traditional CAT(0) metric.

Theorem C (Weak CAT(0)). *The existence of a* CAT(0) *metric is strictly stronger than the existence of a weak asymmetric* CAT(0) *metric in the following sense. There exists a simplicial complex L that admits a weak asymmetric* CAT(0) *metric invariant under all of the orientation-preserving symmetries of the complex, but L does not admit a* CAT(0) *metric invariant under all of its symmetries.*

CHAPTER VI

CONCLUSION

In this dissertation, I have examined the possible piecewise Euclidean metrics on a pair of finite 4-dimensional simplicial complexes to check whether they support symmetry-respecting nonpositively curved metrics of this type. Our ability to test even seemingly simple conditions in high dimensions remains primitive at best. The results in this dissertation and the computer program which accompanies them are contributions to the field of geometric group theory which help to extend our ability to test conditions, such as nonpositive curvature, beyond the low-dimensional cases frequently studied by geometric group theorists. These efforts are important because there are many open conjectures in the field which are likely to be false in general, but true in low dimensions.

In the situations I considered, I used the symmetry-respecting hypothesis to both reduce the dimension from 4 to 3 and to simplify the system of inequalities derived from the simple closed loops in the codimension 2 links. This method of establishing that particular simplicial complexes do not support piecewise Euclidean nonpositively curved metrics was first used by Martin Bridson to show that "outer space" (also known as Culler-Vogtmann space) does not support such a metric [15]. I have shown that the Brady-Krammer complexes of type F_4 and D_4 do not support any symmetry-respecting piecewise Euclidean metrics of nonpositive curvature. Types A_4, B_4 and all of the lower dimensional cases do support such metrics. These results are summarized in Figure 66.

The ordering shown is meant to indicate inclusions among the irreducible Coxeter groups. The Dynkin diagrams of type D_n for $n > 4$, E_6, E_7 and E_8 are lightly crossed out because of the existence of a simplicial subcomplex in the Brady-Krammer

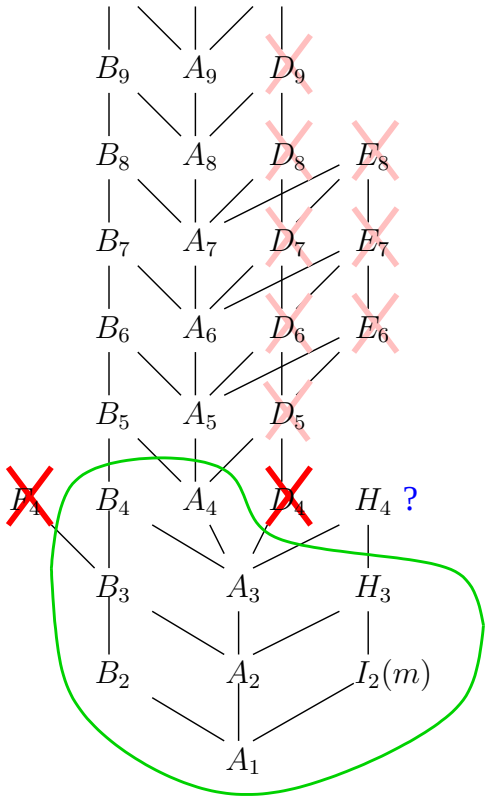


Fig. 66. A summary of curvature results for Artin groups of finite type.

complexes of these types which is isomorphic to the Brady-Krammer complex of type D_4. Although it is not clear that all of symmetries of this type D_4 subcomplex extend to the full complex, it is clear that under suitable mild hypotheses we could restrict the possible metrics of the subcomplex to the ones considered in Chapter V forcing the possible metrics on the whole complex to not be nonpositively curved. Making this observation precise is one direction for further research.

Another obvious direction to pursue is to consider the case of the Brady-Krammer complex of type H_4. The software is available, the linear system of inequalities can be produced and reduced to a simpler form, but because the H_4 Dynkin diagram has no symmetries and the group itself has 60 reflections, the size of this linear system (638

inequalities in 96 variables) makes it difficult to analyze, even with the aid of computer algebra systems. Other techniques need to be developed to deal with situations of this type. Once developed these techniques could also be used to study the class of all piecewise Euclidean metrics on the Brady-Krammer complexes of type D_4 and F_4 (i.e. without the symmetry-respecting hypothesis). One aspect of this research which remains mysterious is the way in which the natural metric is forced upon us in each case. A uniform proof that this is indeed the case, particularly one which explains the reason for this behavior, would be of a great interest. Also of interest would be an explanation, at the group level, which explained the distinction between the groups of type A_4 and B_4 and those of type D_4 and F_4. I plan to pursue all of these avenues of research in the near future.

REFERENCES

[1] E. Artin, *Theorie der zöpfe*, Hamburg Abhandlung **4** (1926), 539–549.

[2] ———, *Theory of braids*, Ann. of Math. (2) **48** (1947), 101–126. MR 8,367a

[3] ———, *The theory of braids*, American Scientist **38** (1950), 112–119. MR 11,377f

[4] D. Bessis, *The dual braid monoid*, Ann. Sci. Ecole Norm. Sup. **36** (2003), no. 5, 647–683, preprint at arXiv:math.GR/0101158 (http://arxiv.org/abs/math.GR/0101158).

[5] Mladen Bestvina, *Non-positively curved aspects of Artin groups of finite type*, Geom. Topol. **3** (1999), 269–302 (electronic). MR 2000h:20079

[6] Nicolas Bourbaki, *Lie groups and Lie algebras. Chapters 4–6*, Elements of Mathematics (Berlin), Springer-Verlag, Berlin, 2002, Translated from the 1968 French original by Andrew Pressley. MR 2003a:17001

[7] Noel Brady and John Crisp, *Two-dimensional Artin groups with* CAT(0) *dimension three*, Proceedings of the Conference on Geometric and Combinatorial Group Theory, Part I (Haifa, 2000), vol. 94, 2002, pp. 185–214. MR 1 950 878

[8] Noel Brady, Jonathan P. McCammond, Bernhard Mühlherr, and Walter D. Neumann, *Rigidity of Coxeter groups and Artin groups*, Proceedings of the Conference on Geometric and Combinatorial Group Theory, Part I (Haifa, 2000), vol. 94, 2002, pp. 91–109. MR 2004b:20052

[9] T. Brady and C. Watt, *$K(\pi, 1)$'s for Artin groups of finite type*, preprint at arXiv:math.GR/0102078 (http://arxiv.org/abs/math.GR/0102078), Feb. 2001.

[10] Thomas Brady, *Artin groups of finite type with three generators*, Michigan Math. J. **47** (2000), no. 2, 313–324. MR 2001j:20056

[11] ———, *A partial order on the symmetric group and new $K(\pi, 1)$'s for the braid groups*, Adv. Math. **161** (2001), no. 1, 20–40. MR 2002k:20066

[12] Thomas Brady and Jonathan P. McCammond, *Three-generator Artin groups of large type are biautomatic*, J. Pure Appl. Algebra **151** (2000), no. 1, 1–9. MR 2001f:20076

[13] Thomas Brady and Colum Watt, *$K(\pi, 1)$'s for Artin groups of finite type*, Proceedings of the Conference on Geometric and Combinatorial Group Theory, Part I (Haifa, 2000), vol. 94, 2002, pp. 225–250. MR 1 950 880

[14] Tom Brady and Jon McCammond, *Artin groups of type A_4 and B_4 are* CAT*(0) groups*, Unpublished, 2004.

[15] Martin R. Bridson, *Geodesics and curvature in metric simplicial complexes*, Group theory from a geometrical viewpoint(Trieste, 1990), World Sci. Publishing, River Edge, NJ, 1991, pp. 373–463.

[16] Martin R. Bridson and André Haefliger, *Metric spaces of non-positive curvature*, Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], vol. 319, Springer-Verlag, Berlin, 1999. MR 2000k:53038

[17] Egbert Brieskorn and Kyoji Saito, *Artin-Gruppen und Coxeter-Gruppen*, Invent. Math. **17** (1972), 245–271. MR 48 #2263

[18] Ruth Charney, *Artin groups of finite type are biautomatic*, Math. Ann. **292** (1992), no. 4, 671–683. MR 93c:20067

[19] ———, *Injectivity of the positive monoid for some infinite type Artin groups*, Geometric group theory down under (Canberra, 1996), de Gruyter, Berlin, 1999, pp. 103–118. MR 2000h:20073

[20] Ruth Charney and Michael W. Davis, *Finite $K(\pi, 1)$s for Artin groups*, Prospects in topology (Princeton, NJ, 1994), Ann. of Math. Stud., vol. 138, Princeton Univ. Press, Princeton, NJ, 1995, pp. 110–124. MR 97a:57001

[21] Ruth Charney, John Meier, and Kim Whittlesey, *Bestvina's normal form complex and the homology of Garside groups*, preprint at arXiv:math.GR/0202228 (http://arxiv.org/abs/math.GT/0301056).

[22] Ruth Charney and David Peifer, *The $K(\pi, 1)$-conjecture for the affine braid groups*, Comment. Math. Helv. **78** (2003), no. 3, 584–600. MR 1 998 395

[23] Arjeh M. Cohen and David B. Wales, *Linearity of Artin groups of finite type*, Israel J. Math. **131** (2002), 101–123. MR 2003j:20062

[24] J. H. Conway and N. J. A. Sloane, *Sphere packings, lattices and groups*, third ed., Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], vol. 290, Springer-Verlag, New York, 1999, With additional contributions by E. Bannai, R. E. Borcherds, J. Leech, S. P. Norton, A. M. Odlyzko, R. A. Parker, L. Queen and B. B. Venkov. MR 2000b:11077

[25] John Crisp, *Injective maps between Artin groups*, Geometric group theory down under (Canberra, 1996), de Gruyter, Berlin, 1999, pp. 119–137. MR 2001b:20064

[26] John Crisp and Luis Paris, *The solution to a conjecture of Tits on the subgroup generated by the squares of the generators of an Artin group*, Invent. Math. **145** (2001), no. 1, 19–36. MR 2002j:20069

[27] M. W. Davis and I. J. Leary, *The l^2-cohomology of Artin groups*, J. London Math. Soc. (2) **68** (2003), no. 2, 493–510. MR 1 994 696

[28] Patrick Dehornoy and Luis Paris, *Gaussian groups and Garside groups, two generalisations of Artin groups*, Proc. London Math. Soc. (3) **79** (1999), no. 3, 569–604. MR 2001f:20061

[29] Pierre Deligne, *Les immeubles des groupes de tresses généralisés*, Invent. Math. **17** (1972), 273–302. MR 54 #10659

[30] François Digne, *On the linearity of Artin braid groups*, J. Algebra **268** (2003), no. 1, 39–57. MR 2 004 479

[31] M Elder and J. McCammond, *CAT(0) is an algorithmic property*, preprint at arXiv:math.GT/0301056 (http://arxiv.org/abs/math.GT/0301056), Jan. 2003.

[32] Murray Elder and Jon McCammond, *Curvature testing in 3-dimensional metric polyhedral complexes*, Experiment. Math. **11** (2002), no. 1, 143–158. MR 2004a:20045

[33] B. Farb, *Relatively hyperbolic groups*, Geom. Funct. Anal. **8** (1998), no. 5, 810–840. MR 99j:20043

[34] The GAP Group, *GAP – Groups, algorithms, and programming, version 4.3*, 2002, (http://www.gap-system.org).

[35] Eddy Godelle, *Parabolic subgroups of Artin groups of type FC*, Pacific J. Math. **208** (2003), no. 2, 243–254. MR 2004c:20062

[36] M. Gromov, *Hyperbolic groups*, Essays in group theory, Math. Sci. Res. Inst. Publ., vol. 8, Springer, New York, 1987, pp. 75–263. MR 89e:20070

[37] Mikhael Gromov, *Infinite groups as geometric objects*, Proceedings of the International Congress of Mathematicians, Vol. 1, 2 (Warsaw, 1983) (Warsaw), PWN, 1984, pp. 385–392. MR 87c:57033

[38] L. C. Grove and C. T. Benson, *Finite reflection groups*, second ed., Graduate Texts in Mathematics, vol. 99, Springer-Verlag, New York, 1985. MR 85m:20001

[39] James E. Humphreys, *Reflection groups and Coxeter groups*, Cambridge Studies in Advanced Mathematics, vol. 29, Cambridge University Press, Cambridge, 1990. MR 92h:20002

[40] Richard Kane, *Reflection groups and invariant theory*, CMS Books in Mathematics/Ouvrages de Mathématiques de la SMC, 5, Springer-Verlag, New York, 2001. MR 2002c:20061

[41] Daan Krammer, *The braid group B_4 is linear*, Invent. Math. **142** (2000), no. 3, 451–486. MR 2001k:20078

[42] ———, *Braid groups are linear*, Ann. of Math. (2) **155** (2002), no. 1, 131–156. MR 2003c:20040

[43] Jonathan P. McCammond, *Noncrossing partitions in surprising locations*, http://www.math.ucsb.edu/

[44] Luis Paris, *On the fundamental group of the complement of a complex hyperplane arrangement*, Arrangements—Tokyo 1998, Adv. Stud. Pure Math., vol. 27, Kinokuniya, Tokyo, 2000, pp. 257–272. MR 2001k:32050

[45] ———, *Artin monoids inject in their groups*, Comment. Math. Helv. **77** (2002), no. 3, 609–637. MR 2003j:20065

APPENDIX A

THE SOFTWARE

A. GAP code

```
############
# coxeter.g
############
#
# created 31 Jan 03
# by Jon McCammond and Woonjung Choi

# streamlined version of the earlier routineswhich
# uses the reflections (stored in reflections.g)
# as input but otherwise avoids using chevie at all.

# The routines can be split into three parts
# 1. generic rountines for arbitrary rank
# 2. testing CAT(0) in the std metric for rank 4
# 3. constructing the 3-complex for rank 4

Print("...loading the coxeter reflections\n");
Read("reflections.g");
Print("\n To begin type something like createCoxeterInfo(\"A\",4);\n");
Print(" where (\"A\",4) can be replaced with any irreducible Coxeter\n");
Print(" type up to rank ",Length(CGTypeList),"\n\n");

# global variables
if not IsBound(type) then type:=[];fi;
if not IsBound(cw) then cw:=();fi;
if not IsBound(w) then w:=[];fi;
if not IsBound(R) then R:=[];fi;
if not IsBound(L) then L:=[];fi;
if not IsBound(len) then len:=[];fi;
if not IsBound(clen) then clen:=[];fi;
if not IsBound(V) then V:=[];fi;
if not IsBound(P) then P:=[];fi;
if not IsBound(PL) then PL:=[];fi;
if not IsBound(c) then c:=[];fi;
if not IsBound(wc) then wc:=();fi;
if not IsBound(paths) then paths:=[];fi;
if not IsBound(col) then col:=[];fi;
if not IsBound(columns) then columns:=[];fi;
if not IsBound(colPatterns) then colPatterns:=[];fi;
if not IsBound(badTri) then badTri:=[];fi;
if not IsBound(badPatterns) then badPatterns:=[];fi;
if not IsBound(verbose) then verbose:=true;fi;
if not IsBound(nPatterns) then nPatterns:=0;fi;
if not IsBound(Diagonals) then Diagonals:=[];fi;
if not IsBound(tetrahedra) then tetrahedra:=[];fi;


###################################
# Part I: Creating the initial Data
###################################
```

```
multiply:=function (L1,L2) # L1,L2 are lists of permutations
    local ans,i;
    ans:=[];
    for i in L1 do Append(ans,i*L2);od;
    return(Set(ans));
end;


kball:=function (L,k) # L is a list of permutations, k is radius
   local ans,i,current,previous,new;
   ans:=[L];
   previous:=[()];
   current:=L;
   for i in [2..k] do
       new:=multiply(current,L);
       SubtractSet(new,Union(current,previous));
       Add(ans,new);
       previous:=current;
       current:=new;
   od;
   return ans;
end;


ksphere:=function (L,k) return kball(L,k)[k];end;


invert:=function(L)
    local i;
    return(List([1..Length(L)],i->L[i]^-1));
end;


lengths:=function(L)
    local i;
    return(List([1..Length(L)],i->Length(L[i])));
end;


levelSets:=function(R,w,n)
    local L,B,l,u,i;
    u:=(n+(n mod 2))/2;
    l:=(n-(n mod 2))/2;
    L:=[];
    B:=kball(R,u);
    for i in [1..u] do L[i]:=Set(B[i]);od;
    IntersectSet(L[u],invert(L[l])*w);
    for i in [1..u-1] do IntersectSet(L[u-i],multiply(R,L[u-i+1]));od;
    for i in [1..l-1] do L[u+i]:=Set(invert(L[l-i])*w);od;
    L[n]:=[w];
    len:=[1];
    Append(len,lengths(L));
    clen:=[1];
    for i in [2..type[2]] do Add(clen,clen[Length(clen)]+len[i]);od;
    V:=[()];
    Append(V,Flat(L));
    return L;
end;


coxeterWord:=function(t,n)
    local i,u,l,r,s,ans;
    u:=(n+(n mod 2))/2;
    l:=(n-(n mod 2))/2;
```

```
    r:=[];s:=[];
    for i in [1..u] do r[i]:=2*i-1;od;
    for i in [1..l] do s[i]:=2*i;od;
    ans:=[];
    if t="D" then
        ans:=[1];
        Append(ans,s);
        Append(ans,List([2..Length(r)],i->r[i]));
    elif t="E" then
        ans:=[1];
        Append(ans,List([2..Length(s)],i->s[i]));
        Append(ans,[2]);
        Append(ans,List([2..Length(r)],i->r[i]));
    else
        ans:=r;
        Append(ans,s);
    fi;
    return(ans);
end;


coxeterElement:=function(L,l)
    local ans,i;
    ans:=();
    for i in l do ans:=ans*L[i];od;
    return(ans);
end;


conjugates:=function(R)
    local i,j;
    return(List([1..Length(R)],i->List([1..Length(R)],j->Position(R,R[j]^R[i]))));
end;


wconjugates:=function(R,w)
    local i;
    return(PermList(List([1..Length(R)],i->Position(R,R[i]^w))));
end;


wconj2:=function(R,w)
    local i;
    return(PermList(List(
       [1..(Length(V)-2-2*Length(R))],i->(Position(V,V[i+Length(R)+1]^w)-(Length(R)+1)))));
end;


createPoset:=function(L,R)
    local i,po,p,q,r;
    po:=[];
    for i in [1..Length(L)] do
        for p in L[i] do
            for r in R do
                q:=r*p;
                if i=1  then
                    if q=() then
                        Add(po,[1,Position(V,p),Position(R,r)]);
                    fi;
                else
                    if (q in L[i-1]) then
                        Add(po,[Position(V,q),Position(V,p),Position(R,r)]);
                    fi;
                fi;
```

```
            od;
        od;
    od;
    return(Set(po));
end;


stripLabels:=function(P)
    local po,p;
    po:=[];
    for p in P do Add(po,[p[1],p[2]]);od;
    return(po);
end;


################################################
# This is main initial function.  It creates the
# posets and all their associated information.
################################################
createCoxeterInfo:=function(l,n)
    type:=[l,n];
    Print("\ntype = Coxeter group (\"",l,"\",",n,")\n");

          R:=CGReflections(l,n);
    Print("R    = ",Length(R)," reflections of W\n");

    cw:=coxeterWord(l,n);
    w:=coxeterElement(R,cw);
    Print("w    = coxeter element (",cw,")\n");
    Print("\n");

    c:=conjugates(R);
    wc:=wconjugates(R,w);
    Print("c  = conjugacy table for R\n");
    Print("wc = conjugates classes for R rel w\n");
    Print("\n");

    L:=levelSets(R,w,n);
    Print("L  = permutations below w (",len,")\n");
    Print("V  = ",Length(V)," permutations listed in L\n\n");

    PL:=createPoset(L,R);
    P:=stripLabels(PL);
    Print("P  = poset structure for L\n");
    Print("PL = poset structure for L with labels\n\n");

    Print("Other information is stored in len, clen, and cw --but \n");
    Print("these are mostly for internal use.\n\n");

    Print("  To calculate the column structure, type createColumns();\n\n");

    if n=4 then
        Print("  Since this group is rank 4, try testGroup(); to see whether\n");
        Print("  the Brady-Krammer complex for the Artin group of this type\n");
        Print("  is CAT(0) using the standard coxeter shape metric.\n");
        Print("\n");
    fi;
end;


######################################################################
# Part II: Testing for CAT(0) in rank 4 (using the standard metric)
######################################################################
caps:=function(po,L)
```

```
    local i,j,ans,add;
    ans:=[];
    for i in po do
        if i[1]=L[1] then
            add:=true;
            for j in [2..Length(L)] do
                if not [L[j],i[2]] in po then add:=false;fi;
            od;
            if add then Add(ans,i[2]);fi;
        fi;
    od;
    return(ans);
end;


coverPairs:=function(po,n)
    local i,j,l,ca,c,cp,s;
    cp:=[];s:=[];
    for i in po do if i[1]=n then Add(s,i[2]);fi;od;
    s:=Set(s);
    l:=Length(s);
    for i in [1..l-1] do
        for j in [i+1..l] do
            ca:=caps(po,[s[i],s[j]]);
            if not ca=[] then
                for c in ca do Add(cp,[s[i],s[j],c]);od;
            fi;
        od;
    od;
    return(Set(cp));
end;


findTris:=function(L)
    local i,j,k,goodtri,badtri,l,e,f;
    goodtri:=[];
    badtri:=[];
    l:=Length(L);
    for i in [1..l-1] do
        for j in [i+1..l] do
            if L[i][2]=L[j][1] then
                for k in L do
                    if (k[1]=L[i][1]) and (k[2]=L[j][2]) then
                        e:=Set([L[i][1],L[i][2],L[j][2]]);
                        f:=Set([L[i][3],L[j][3],k[3]]);
                        if Length(f)=3 then Add(badtri,[e,f]);fi;
                    fi;
                od;
            fi;
        od;
    od;
    return(Set(badtri));
end;


testTri:=function(Tr,po)
    local i,bad,new;
    bad:=[];
    for i in Tr do
        if caps(po,i[2])=[] then
            new:=[Position(R,V[i[1][1]]),Position(R,V[i[1][2]]),Position(R,V[i[1][3]])];
            Add(bad,Set(new));
        fi;
    od;
```

```
    return(Set(bad));
end;

reflectionType:=function(n) return(Set(Cycle(wc,n))[1]);end;


pattern:=function(L)
    local i;
    return(List([1..Length(L)],i->reflectionType(L[i])));
end;


#####
# minCycle is to find the minimal cycle of the given one up to numeric order.
# minCycle2 has the position of the 1st coordinate in the original cycle.
# ex) gap> minCycle2([3,2,7,1,3]);
#     [ 4, [ 1, 3, 3, 2, 7 ] ]
#####
minCycle:=function(L)
    local i,j,l,LL,m;
    l:=Length(L);
    LL:=List([1..l],i->L[i]);
    for i in [1..l] do LL[i+l]:=L[i];od;
    m:=0;
    for i in [1..l] do
        j:=1;
        while LL[j+m]=LL[j+i] do
            j:=j+1;
            if j>l then return(List([1..l],i->LL[m+i]));fi;
        od;
        if LL[j+m]>LL[j+i] then m:=i;fi;
    od;
    return(List([1..l],i->LL[m+i]));
end;


minCycle2:=function(L)
    local i,j,l,LL,m;
    l:=Length(L);
    LL:=List([1..l],i->L[i]);
    for i in [1..l] do LL[i+l]:=L[i];od;
    m:=0;
    for i in [1..l] do
        j:=1;
        while LL[j+m]=LL[j+i] do
            j:=j+1;
            if j>l then return(List([1..l],i->LL[m+i]));fi;
        od;
        if LL[j+m]>LL[j+i] then m:=i;fi;
    od;
    return([m+1,List([1..l],i->LL[m+i])]);
end;


##########################################
# This is the main function to test CAT(0)
# for 4 generator Artin groups
##########################################
testGroup:=function()
    local i;
    if not type[2]=4 then
        Print("\nThis is not rank 4\n\n");
        return;
```

```
    fi;
    badTri:=testTri(findTris(coverPairs(P,1)),P);
    badPatterns:=[];
    for i in badTri do Add(badPatterns,minCycle(pattern(i)));od;
    badPatterns:=Set(badPatterns);
    if badTri=[] then
        Print("\nThe type (",type[1],",4) Brady-Krammer complex is CAT(0)\n");
        Print("using the standard metric.\n\n");
        return;
380 else
        Print("\n  The type (",type[1],",4) Brady-Krammer complex is not CAT(0)\n");
        Print("  using the standard metric because some triples of reflections\n");
        Print("  give rise to short geodesic cycles in the unique vertex link\n\n");
        Print("  There are ",Length(badTri)," bad triples of reflections\n");
        Print("  There are ",Length(badPatterns)," bad patterns of reflections\n\n");
        Print("  To see the bad triples type badTri;\n");
        Print("  To see the bad patterns type badPatterns;\n\n");
    fi;
end;
390


###################################
# Part III: Create Paths and Columns
###################################


#####
# createVertexPaths(Length(V)) generates all paths with vertex names such as
400 #       [[1,8,15,38,50],...]
# createPathLabels(Length(V)) generates all path with reflection names such as
#       [[1,2,3,7],[1,2,4,3],...]
# createVertexPaths(Length(V))[i] & createPathLabels(Length(V))[i] DO NOT match.
#####

createVertexPaths:=function(n)
    local lev,ans,i;
    if n<=clen[2] then return([[1,n]]);fi;
    lev:=type[2];
410 while n<=clen[lev] do lev:=lev-1;od;
    ans:=[];
    for i in [clen[lev-1]+1..clen[lev]] do
        if [i,n] in P then Append(ans,createVertexPaths(i));fi;
    od;
    for i in [1..Length(ans)] do Add(ans[i],n);od;
    return ans;
end;

createPathLabels:=function(n)
420 local i,j,ans,p;
    paths:=createVertexPaths(n);
    ans:=[];
    for i in [1..Length(paths)] do
        ans[i]:=[];
        for j in [2..Length(paths[i])] do
            Add(ans[i],Position(R,V[paths[i][j-1]]^-1*V[paths[i][j]]));
        od;
    od;
    return Set(ans);
430 end;


# This creates the vertex paths with the reflections
createPathLabels2:=function(n)
```

```
    local i,j,ans,p;
    paths:=createVertexPaths(n);
    ans:=[];
    for i in [1..Length(paths)] do
        ans[i]:=[];
        ans[i][1]:=paths[i];
        ans[i][2]:=[];
        for j in [2..Length(paths[i])] do
            Add(ans[i][2],Position(R,V[paths[i][j-1]]^-1*V[paths[i][j]]));
        od;
    od;
    return Set(ans);
end;


shiftCol:=function(s)
    local i,j,ans;
    ans:=List([2..Length(s)],j->s[j]);
    Add(ans,s[1]^wc);
    return(ans);
end;


columnNumbers:=function(L)
    local i;
    return(List([1..Length(L)],i->paths[L[i]][1]));
end;


#####################################
# This is the function which finds the
# columns in the Brady-Krammer complex
#####################################
createColumns:=function()
    local shift,ans,cy,i;
    paths:=createPathLabels(Length(V));
    shift:=List([1..Length(paths)],i->Position(paths,shiftCol(paths[i])));
    col:=PermList(shift);
    cy:=CycleStructurePerm(col);
    columns:=[];
    for i in [1..Length(paths)] do Add(columns,minCycle(columnNumbers(Cycle(col,i))));od;
    columns:=Set(columns);
    colPatterns:=[];
    for i in columns do Add(colPatterns,minCycle(pattern(i)));od;
    colPatterns:=Set(colPatterns);
    if verbose then
    Print("\n");
    Print(" There are ",Length(paths)," simplices\n");
    for i in [1..Length(cy)] do
        if IsBound(cy[i]) then
            if cy[i]>1 then
                Print(" There are ",cy[i]," columns each containing ",
                      i+1," simplices\n");
            else
                Print(" There is 1 column containing ",i+1," simplices\n");
            fi;
        fi;
    od;
    Print(" There are ",Length(colPatterns)," column patterns\n");
    Print("\nType paths; to see the reflections creating the simplices\n");
    Print("Type columns; to see the sequences creating the columns\n");
    Print("Type colPatterns; to see the list of column patterns\n");
    Print("\n");
```

```
    Print("If this type has diagram symmetries, the column patterns can\n");
    Print("be simplified by typing simplifyPatterns(s); where s is a\n");
    Print("list of length ",type[2]," that the i-th generater should\n");
    Print("be replaced with.  The list of patterns returned will be\n");
    Print("cyclically minimal and without repetitions.\n\n");
    fi;
end;


simplify:=function(L,s)
    local i,j,ans;
    ans:=List([1..Length(L)],i->[]);
    for i in [1..Length(L)] do
        for j in L[i] do
            Add(ans[i],s[j]);
        od;
    od;
    return ans;
end;


simplifyPatterns:=function(s)
    local i,ans;
    ans:=simplify(colPatterns,s);
    for i in [1..Length(ans)] do
        ans[i]:=minCycle(ans[i]);
    od;
    return(Set(ans));
end;


############################
# Part VI create tetrahedra
############################

### z is the diagram symmetries.
z:=function(t)
    local ans;
    if t="A" then
        ans:=[1,2,2,1];
    elif t="B" then
        ans:=[1,2,3,4];
    elif t="D" then
        ans:=[1,1,2,1];
    elif t="F" then
        ans:=[1,2,2,1];
    elif t="H" then
        ans:=[1,2,3,4];
    fi;
    return(ans);
end;


## simpPatterns generates minimal cycles of the given cycles of repeating 4 digits
#  with the original position of the first coordinate of the output.
#  ex : gap> simpPatterns([[2,3,1,4,2,3,1,4],[2,2,1,4,2,2,1,4]]);
##     " [ [ 3, [ 1, 4, 2, 3, 1, 4, 2, 3 ] ], [ 3, [ 1, 4, 2, 2, 1, 4, 2, 2 ] ]"
simpPatterns:=function(b)
    local i,ans;
    ans:=List([1..Length(b)],i->[]);
    for i in [1..Length(b)] do
        ans[i]:=[minCycle2([b[i][1],b[i][2],b[i][3],b[i][4]])[1],minCycle(b[i])];
    od;
```

```
    return(ans);
end;


### CreateDiagonalSets gives list of sets for diagonals of each columns
# this is a subloop for createCapset.
createDiagonalSets:=function()
    local i,j,k,a,b,c,d,l;
    verbose:=false;
    createColumns();
    verbose:=true;
    a:=List([1..Length(columns)],i->pattern(columns[i]));
    b:=simplify(a,z(type[1]));
    c:=simpPatterns(b);
    d:=List([1..2*(Length(columns))],i->[]); # couples for the diagonal in coulumn
    for i in [1..Length(columns)] do
        if (Length(columns[i]) mod 2)=0 then
            l:=Length(columns[i])/2;
            if c[i][1] in [1,3] then
                for j in [1..l] do
                    Add(d[2*i-1],[columns[i][2*j-1],columns[i][2*j]]);
                od;
                for k in [1..l-1] do
                    Add(d[2*i],[columns[i][2*k],columns[i][2*k+1]]);
                od;
                Add(d[2*i],[columns[i][Length(columns[i])],columns[i][1]]);
            fi;
            if c[i][1] in [2,4] then
                for j in [1..l] do
                    Add(d[2*i],[columns[i][2*j-1],columns[i][2*j]]);
                od;
                for k in [1..l-1] do
                    Add(d[2*i-1],[columns[i][2*k],columns[i][2*k+1]]);
                od;
                Add(d[2*i-1],[columns[i][Length(columns[i])],columns[i][1]]);
            fi;
        elif (Length(columns[i]) mod 2)=1 then
            l:=Length(columns[i]);
            for j in [1..l-1] do
                Add(d[2*i-1],[columns[i][j],columns[i][j+1]]);
            od;
            Add(d[2*i-1],[columns[i][l],columns[i][1]]);
            for k in d[2*i-1] do
                Add(d[2*i],k);
            od;
        fi;
    od;
    return([a,b,c,d]);
end;


### this creates capset which is the list whose ith coordinate is every numbers j such that
# d[i] cap d[j] is nonempty.
# this is a subloop for createDiagonals.
createCapset:=function()
     local d,i,j,capset;
    d:=createDiagonalSets()[4];
    capset:=List([1..2*(Length(columns))],i->[]);
    for i in [1..2*(Length(columns))] do
         for j in [1..2*(Length(columns))] do
             if Length(Intersection(d[i],d[j]))>0 then
                Add(capset[i],j);
             fi;
```

```
            od;
        od;
        return(capset);
    end;

    # this function creates diagonals based on the intersection of
    # their diagonals without regarding column types.
    # this is a subloop for createDiagonals.
    createDia:=function()
         local i,j,k,
               dia,
               m,
               capset,
               A,B,C,
               checked;
        capset:=createCapset();
        dia:=List([1..2*(Length(columns))],i->-1);
        A:=[1..Length(capset)];
        checked:=List([1..Length(capset)], i->0);   # not checked;
        m:=5;
        for i in A do
            if checked[i] = 0 then  # not-checked, thus do it.
                B:=capset[i];
                for j in capset[i] do
                    C:=UnionSet(B,capset[j]);
                    while Length(Difference(C,B))>0 do
                        for k in Difference(C,B) do
                            B:=C;
                            C:=UnionSet(B,capset[k]);
                         od;
                    od;
                od;
                for j in B do
                    checked[j]:=1;   # checked and assigned
                    dia[j]:=m;
                od;
                m:=m+1;
            fi;
        od;
        return(dia);
    end;


    # createTypeSets creates list of length L which maps each coordinate of L
    # to the coresponding element of M.
    # this is a subloop for createDiagonals.
    createTypeSets:=function(M,L)
       local se,i,j;
        se:=List([1..Length(L)],i->[]);
        for i in [1..Length(columns)] do
            for j in [1..Length(L)] do
                if M[i]=L[j] then
                   Add(se[j],i);
                fi;
            od;
        od;
        return(se);
    end;


    # this creates diagonals for each columns.
    createDiagonals:=function()
```

```
local dia,simple,diagonals,minE,minF,minI,minJ,diaJ,minK,diaK,minII,
      diagonals1,minN,diaset,diaset1,idx,
      a,b,c,d,e,f,i,j,k,n,p,q,r,s,u,x,y,l,m,t,
      A,B,C,D,E,F,H,I,J,K,L,M,N,O;

dia:=createDia();
simple:=simplifyPatterns(z(type[1]));
c:=createDiagonalSets()[3];               # columns in terms of generators with shift.
diaset:=createDiagonalSets()[4];          # couples for each diagonal of each columns.
diagonals:=StructuralCopy(dia);
A:=List([1..Length(columns)],i->0);       # index list for columns as patterns.
for i in [1..Length(columns)] do
     for j in [1..Length(simple)] do
        if simple[j]=c[i][2] then
           A[i]:=j;
        fi;
     od;
od;
B:=[];C:=[];                              # B: columns with more than one generator.
for k in [1..Length(simple)] do       # C: columns with one generator.
     if Length(Set(simple[k]))>1 then
        Add(B,k);
     fi;
od;
Append(C,Difference([1..Length(simple)],B));
idx := List([1..Length(columns)], i->0);
H:=createTypeSets(A,C);                   # columns for each type with one generator.
for x in [1..Length(H)] do
    t:=1;
    while t>0 do
        diagonals1:=StructuralCopy(diagonals);  # for comparison.
        I:=[];                                  # I: for a tetra with one diagonal value.
        for y in H[x] do
            if diagonals1[2*y-1]=diagonals1[2*y] then
               Add(I,diagonals1[2*y]);
            fi;
        od;
        if Length(I)>0 then        # if I not empty, we get all same value for the type.
            minI:=Minimum(I);
            for l in H[x] do
                for m in [1..Length(dia)] do
                    if diagonals[m]=diagonals1[2*l-1] then
                       diagonals[m]:=minI;
                    fi;
                    if diagonals[m]=diagonals1[2*l] then
                       diagonals[m]:=minI;
                    fi;
                od;
            od;
        fi;
        if Length(I)=0 then
            for a in H[x] do         # finding same values and matching the pairs.
            J:=[];K:=[];
                for b in H[x] do
                    if diagonals[2*a-1]=diagonals[2*b-1] then
                       Append(J,[2*a,2*b]);
                    fi;
                    if diagonals[2*a-1]=diagonals[2*b] then
                       Append(J,[2*a,2*b-1]);
                    fi;
                    if Length(J)>2 then
                         diaJ:=[];
                         for f in J do
```

```
                                Add(diaJ,diagonals[f]);
                            od;
                            minJ:=Minimum(diaJ);
                            for d in J do
                                for e in [1..Length(dia)] do
                                    if diagonals[e]=diagonals1[d] then
                                       diagonals[e]:=minJ;
                                    fi;
                                od;
                            od;
                            J:=[J[1],J[2]];
                        fi;
                        diagonals1:=StructuralCopy(diagonals);
                        if diagonals[2*a]=diagonals[2*b-1] then
                           Append(K,[2*a-1,2*b]);
                        fi;
                        if diagonals[2*a]=diagonals[2*b] then
                           Append(K,[2*a-1,2*b-1]);
                        fi;
                        if Length(K)>2 then
                            diaK:=[];
                            for f in K do
                                Add(diaK,diagonals[f]);
                            od;
                            minK:=Minimum(diaK);
                            for d in K do
                                for e in [1..Length(dia)] do
                                    if diagonals[e]=diagonals1[d] then
                                       diagonals[e]:=minK;
                                    fi;
                                od;
                            od;
                            K:=[K[1],K[2]];
                        fi;
                        diagonals1:=StructuralCopy(diagonals);
                    od;
                od;
            fi;

            if diagonals=diagonals1 then
               t:=0;
            fi;
            for i in [1..Length(H)] do
                L:=[];M:=[];
                for j in H[i] do
                    L:=[diagonals[2*j-1],diagonals[2*j]];
                    Add(M,Length(Set(L)));
                od;
                if Maximum(M)-Minimum(M)>0 then   # in case one pattern has more
                    t:=1;                         # than one type set of diagonals.
                fi;
            od;
        od;
        N:=[];                          # in case same type of tetra with diff. order
        for i in H[x] do                # of diagonals, we can change them.
            Add(N,diagonals[2*i-1]);
        od;
        if Length(Set(N))>1 then
            minN:=Minimum(N);
            O:=List([1..Length(H[x])],i->0);
            for i in H[x] do
                if diagonals[2*i-1]<>minN then
                   O[Position(H[x],i)]:=1;
                fi;
```

```
                od;
                diaset1:=StructuralCopy(diaset);
                for j in [1..Length(O)] do
                    if O[j]=1 then
820                     diaset[2*H[x][j]-1]:=diaset1[2*H[x][j]];
                        diaset[2*H[x][j]]:=diaset1[2*H[x][j]-1];
                        diagonals[2*H[x][j]-1]:=diagonals1[2*H[x][j]];
                        diagonals[2*H[x][j]]:=diagonals1[2*H[x][j]-1];
                        idx[H[x][j]]:=1;
                    fi;
                od;
            fi;
        od;
        D:=createTypeSets(A,B);        # columns with more than one generator.
830     for n in [1..Length(D)] do
            E:=[];
            F:=[];
            for p in D[n] do
                Add(E,diagonals[2*p-1]);
                Add(F,diagonals[2*p]);
            od;
            if Length(Intersection(E,F))>0 then
                 Append(E,F);
                 minE:=Minimum(E);
840              for q in E do
                    for r in [1..Length(dia)] do
                       if diagonals[r]=q then
                          diagonals[r]:=minE;
                       fi;
                    od;
                 od;
            elif Length(Intersection(E,F))=0 then
                 minE:=Minimum(E);
                 minF:=Minimum(F);
850              for q in E do
                     for r in [1..Length(dia)] do
                        if diagonals[r]=q then
                           diagonals[r]:=minE;
                        fi;
                     od;
                 od;
                 for s in F do
                     for u in [1..Length(dia)] do
                        if diagonals[u]=s then
860                        diagonals[u]:=minF;
                        fi;
                     od;
                 od;
            fi;
        od;
        for i in [1..Length(columns)] do
            if diagonals[2*i-1]<>diagonals[2*i] and idx[i]=1 then
                c[i][1]:=2;
            fi;
870     od;
        return([c,A,B,C,D,H,diagonals,diaset,idx]);
    end;


    # columns with diagonals and tetrahedra with all labels.
    createSpces:=function()
        local c,diag, diagonals,A,i,j,k,l,
              ntetra, nDiag;
        diag := createDiagonals();
```

```
    c := diag[1];
    diagonals := diag[7];

    A:=List([1..Length(columns)],i->[]);
    for i in [1..Length(columns)] do
       for j in [1..4] do
           A[i][j]:=c[i][2][j];
       od;
       for k in [5,6] do
           A[i][k]:=diagonals[(2*i)-6+k];
       od;
    od;
    tetrahedra := [];
    ntetra:=Length(Set(diag[2]));
    nDiag := Length(diag[2]);
    for l in [1..ntetra] do
        # find the first occurence of l in diag[2]
        for k in [1..nDiag] do
            if l = diag[2][k] then
                break;
            fi;
        od;
        Add(tetrahedra, A[k]);
    od;
    return([A,tetrahedra]);
end;


################
#createNodes1()[1] is
#nodes to check in rank 1 in V.
#createNodes1()[2] is representatives in numbers
#which apear in z(type[1]).
#i.e. createNodes1()[2] for F_4 is [1,2],
#for H_4 is [1,2,3,4].
#createNodes1()[3][i] is standard generators
#corresponding to createNodes1()[2][i].
#createNodes1()[4] is corresponding nodes in V
#to createNodes1()[3].
################

createNodes1:=function()
    local A,B,C,D,E,N1,
          i,j;
    A:=Set(z(type[1]));
    B:=[];
    for i in A do
        Add(B,Position(z(type[1]),i));
    od;
    D:=List([1..Length(B)],i->[]);
    for i in A do
        for j in [1..4] do
            if z(type[1])[j]=i then
                Add(D[i],j);
            fi;
        od;
    od;
    E:=List([1..Length(D)],i->[]);
    for i in [1..Length(D)] do
        for j in [1..Length(D[i])] do
            E[i][j]:=Position(V,R[D[i][j]]);
        od;
    od;
    N1:=[];
```

```
    C:=[];
    for j in B do
        Add(C,Position(V,R[j]));
    od;
    return([C,B,D,E]);
end;

############
#basics() is conjugates for each standard generators in R.
#basics()[2] provides info that
# each standard generator belongs to which conj.
############

basics:=function()
    local a,A,i,j;
    a:=Cycles(wconjugates(R,w),[1..Length(R)]);
    A:=[0,0,0,0];
    for i in [1..4] do
        for j in [1..Length(a)] do
            if i in a[j] then A[i]:=j; fi;
        od;
    od;
    return([a,A]);
end;

############
#genSet() gives elements in same type of generators in R
############

genSet:=function()
    local a,A,D,i,j,k,cyc;
    D:=createNodes1()[3];
    a:=basics()[1];
    A:=basics()[2];
    cyc:=List([1..Length(D)],i->[]);
    for i in [1..Length(D)] do
        for j in D[i] do
            for k in a[A[j]] do
                Add(cyc[i],k);
            od;
        od;
    od;
    return(cyc);
end;

##########
#genSetNodes() gives nodes in types in V for nodes in rank 1 and 3.
##########

genSetNodes:=function()
    local cyc,nodesSet1,nodesSet2,i,j;
    cyc:=genSet();
    nodesSet1:=StructuralCopy(cyc);
    nodesSet2:=StructuralCopy(cyc);
    for i in [1..Length(cyc)] do
        for j in [1..Length(cyc[i])] do
            nodesSet1[i][j]:=Position(V,R[cyc[i][j]]);
        od;
    od;
    for i in [1..Length(cyc)] do
        for j in [1..Length(cyc[i])] do
```

```
            nodesSet2[i][j]:=Position(V,w*R[cyc[i][j]]);
        od;
    od;
    return([nodesSet1,nodesSet2]);
end;


# subloop for nodes2
mutableCopy:=function(obj)
    local mcp, i;

    mcp := [];
    if IsList(obj) then
        for i in [1..Length(obj)] do
            mcp[i] := mutableCopy(obj[i]);
        od;
    else
        mcp := obj;
    fi;

    return(mcp);
end;


# subloop for createNodes2
nodes2:=function()
    local a,b,c,i,j,A;
    a:=wconj2(R,w);
    A:=Cycles(a,[1..(Length(V)-2-(2*(Length(R))))]);
    b:=mutableCopy(A);
    for i in [1..Length(A)] do
        for j in [1..Length(A[i])] do
            b[i][j]:=A[i][j]+Length(R)+1;
        od;
    od;
    c:=[];
    for i in [1..Length(b)] do
        Add(c,b[i][1]);
    od;
    return([b,c]);
end;


createNodes2:=function()
    local N,A,B,C,
          i,j,k;
    N:=nodes2()[2];
    C:=[];
    for i in N do
        A:=[]; B:=[];
        for j in [1..(Length(P)-2*Length(R))/2] do
            if i=P[j+Length(R)][2] then
                Add(A,P[j+Length(R)][1]);
            fi;
        od;
        for k in [1..(Length(P)-2*Length(R))/2] do
            if i=P[Length(P)/2+k][1] then
                Add(B,P[Length(P)/2+k][2]);
            fi;
        od;
        C[Position(N,i)]:=[A,B];
    od;
    return([N,C]);
end;
```

```
elimNodes :=function()
    local nodes,edges,nodesSet1,nodesSet2,
          edgetypes,
          typeSet,typeSet2,typeSet3,
          Nodes2,
          i,j,k,l,numGen,gptype;
    gptype := type[1];
    numGen := Length(Set(z(gptype)));
    nodes:=createNodes2()[1];
    edges:=createNodes2()[2];
    nodesSet1:=genSetNodes()[1];
    nodesSet2:=genSetNodes()[2];
    edgetypes:=[];
    for i in [1..Length(nodes)] do
        if gptype in ["A","D","F"] then
            edgetypes[i] := [[0,0],[0,0]];
        elif gptype in ["B","H"] then
            edgetypes[i]:= [[0,0,0,0],[0,0,0,0]];
        fi;
    od;
    for i in [1..Length(nodes)] do
        for j in [1,2] do
            for l in edges[i][j] do
                for k in [1..numGen] do
                    if l in nodesSet1[k] then
                    edgetypes[i][1][k]:=edgetypes[i][1][k]+1;
                    elif l in nodesSet2[k] then
                    edgetypes[i][2][k]:=edgetypes[i][2][k]+1;
                    fi;
                od;
            od;
        od;
    od;
    typeSet:=Set(edgetypes);
    typeSet2:=[];
    for i in [1..Length(typeSet)] do
        Add(typeSet2,Set(typeSet[i]));
    od;
    typeSet3:=Set(typeSet2);
    Nodes2:=[];
    for i in [1..Length(typeSet3)] do
        if Length(typeSet3[i])=2 then
            Add(Nodes2,nodes[Position(edgetypes,typeSet3[i])]);
        elif Length(typeSet3[i])=1 then
            Add(Nodes2,nodes[Position(edgetypes,[typeSet3[i][1],typeSet3[i][1]])]);
        fi;
    od;
    return([edgetypes,typeSet,typeSet2,typeSet3,Nodes2]);
end;


nodesCheck:=function()
    return(Concatenation(createNodes1()[1],elimNodes()[5]));
end;


#
# find all nodes which are connected to 's' in a graph, 'graph',
#    and returns nodes in the first element (= b)
#    and their locations defined in 'graph' in the second element (= idx_graph)
```

```
#
findEdges := function(s, graph)
    local n, i, b, idx_graph;
    n := Length(graph[1]);  # edges
    b := [];
    idx_graph := [];
    for i in [1..n] do
        if graph[2][i] = 0 then  # not used
            if graph[1][i][1] = s then
                Add(b, graph[1][i][2]);
                Add(idx_graph, i);
            elif graph[1][i][2] = s then
                Add(b, graph[1][i][1]);
                Add(idx_graph, i);
            fi;
        fi;
    od;
    return([b, idx_graph]);
end;


#
# as a recursive function, this will find all loops containing 'path'
#    and store them in 'paths'.
#
# 'path' is a collection of nodes traveled and can be just a starting node.
#
findClosedLoop := function(path, paths, graph, nVertices)
    local s_edges, n, i, ret, npath;

    # traveled too far, no loop.
    if nVertices+1 = Length(path) then
        return(-1);
    fi;

    # traveled so far
    npath := Length(path);

    # next nodes available -> edges
    s_edges := findEdges(path[npath], graph);
    n := Length(s_edges[1]);
    for i in [1..n] do

        # move forward a step
        Add(path, s_edges[1][i]);

        # and see if a loop is found
        if path[1] = s_edges[1][i] then   # loop found
            Append(paths, [path]);
            return(0);
        else
            graph[2][s_edges[2][i]] := 1;  # used
            ret := findClosedLoop(path, paths, graph, nVertices);
            graph[2][s_edges[2][i]] := 0;  # restore
        fi;

        # move backward a step (to its original location)
        path := path{[1..npath]};
    od;

    return(-1);
end;
```

```
findClosedLoop2 := function(path, paths, graph, nVertices)
    local s_edges, n, i, ret, npath;

    # traveled too far, no loop.
    if nVertices+1 = Length(path) then
        return(-1);
    fi;


    # traveled so far
    npath := Length(path);

    # next nodes available -> edges
    s_edges := findEdges(path[npath], graph);
    n := Length(s_edges[1]);
    for i in [1..n] do

        # move forward a step
        Add(path, s_edges[1][i]);

        # and see if a loop is found
        if path[1] = s_edges[1][i] then   # loop found
            Append(paths, [path]);
            return(0);
        elif npath+1 > Length(Set(path)) then
            return(-1);
        else
            graph[2][s_edges[2][i]] := 1;  # used
            ret := findClosedLoop2(path, paths, graph, nVertices);
            graph[2][s_edges[2][i]] := 0;  # restore
        fi;

        # move backward a step (to its original location)
        path := path{[1..npath]};
    od;

    return(-1);
end;


#
# find All Closed Loops from a given graph
#   need to simplify using minCycle...
#
findAllClosedLoops := function(graph)
    local i, path, paths, nVertices, vertices, graph_flag, grp, nGrp;

    grp := graph[1];  # only graph
    nGrp := Length(grp);

    graph_flag := List([1..nGrp], i->0);  # traveld or not

    vertices := Set(Flat(grp));    # all vertices
    nVertices := Length(vertices);
    vertices := Set(Flat(grp{[1..nGrp]}{[1]}));  # check only a side of edge

    paths := [];
    for i in vertices do
        findClosedLoop2([i], paths, [grp, graph_flag], nVertices+1);
    od;

    return paths;
end;
```

```
# subloop for findGraphForNode.
# pathLabelsToGraph will give a graph 'graphs' for the specific node and
# find spces for all edge in the graph 'spcs'
pathLabelsToGraph := function(pathLabels, nodeNum, numGen)
    local i, n, graphs, spcs;

    n := Length(pathLabels);
    if (2 <= nodeNum) and (nodeNum <= numGen+1) then  # in rank one
        graphs := [];
        spcs := [];
        for i in [1..n] do
            if pathLabels[i][1][2] = nodeNum then
                Add(graphs, pathLabels[i][1]{[3,4]});
                Add(spcs, pathLabels[i][2]);
            fi;
        od;
    else  # in rank two
        graphs := [];
        spcs := [];
        for i in [1..n] do
            if pathLabels[i][1][3] = nodeNum then
                Add(graphs, pathLabels[i][1]{[2,4]});
                Add(spcs, pathLabels[i][2]);
            fi;
        od;
    fi;

    return( [graphs, spcs] );
end;


#
# find a graph for a node
#
findGraphForNode := function(node)
    return(pathLabelsToGraph(createPathLabels2(Length(V)), node, Length(R)));
end;


#
# find all closed loops for a given node
#    simply find a graph for the node and call findAllClosedLoops();
#
findAllClosedLoopsForNode := function(node)
    return(findAllClosedLoops(findGraphForNode(node)));
end;


#
# simpify loops by removing repetions and only taking simple loops
#
simplifyLoops := function(loops)
    local nLoops, minLoops, simplifiedLoops, i, k;

    nLoops := Length(loops);

    minLoops := [];
    for i in [1..nLoops] do

        # breaking loops by removing the closing vertice
        k := Length(loops[i])-1;
```

```
            Add(minLoops, minCycle(loops[i]{[1..k]}));
        od;
        minLoops := Set(minLoops);   # no repeated loops

        nLoops := Length(minLoops);
        simplifiedLoops := [];
        for i in [1..nLoops] do
            # find only simple loops
            if Length(minLoops[i]) = Length(Set(minLoops[i])) then
                Add(minLoops[i], minLoops[i][1]);  # closing back the loop
                Add(simplifiedLoops, minLoops[i]);
            fi;
        od;

        return simplifiedLoops;
    end;

    #
    #
    #
    findAllSimpleClosedLoopsForNode := function(node)
        return(simplifyLoops(findAllClosedLoopsForNode(node)));
    end;


    #
    # this is to find a match of Simplex responding to
    #    each edge of a link in a column and its location
    #
    findSubMatch:=function(spx, dblCol)
        local nspx, ndblCol, i, j, matched;

        nspx := Length(spx);
        ndblCol := Length(dblCol);

        for i in [1..(ndblCol-nspx+1)] do
            if spx[1] = dblCol[i] then   # first match
                matched := 1;
                for j in [2..nspx] do    #  complete match
                    if spx[j] <> dblCol[i+j-1] then  # match broken
                        matched := 0;
                        break;
                    fi;
                od;
                if matched = 1 then
                    return (i);
                fi;
            fi;
        od;
        return (0);  # not found
    end;

    #
    # find spx from list of columns
    #   and return which column and its location in the column
    #
    findMatch:=function(spx, cols)
        local nCols, i, dblCol, pos;

        nCols := Length(cols);
        for i in [1..nCols] do
            dblCol := Flat([cols[i], cols[i]]);  # makes double column
            pos := findSubMatch(spx, dblCol);
```

```
        # pos := PositionSublist(dblCol, spx);
        if pos > 0 then  # found
            return([i, pos]);
        fi;
    od;
end;

#
# find label (=[type, location]) for each edge
#
findEdgeLabel:=function(edge, graphs, cols)
    local i, col_loc, diag, spx, pos;

    # find spx corresponding to edge in graphs
    for i in [1..Length(graphs[1])] do
        if Set(edge) = Set(graphs[1][i]) then
            spx := graphs[2][i];
        fi;
    od;

    # findMatch -> (col, loc)
    col_loc := findMatch(spx, cols);  # col_loc[1] = col, col_loc[2] = location

    diag := createDiagonals();

    pos := (1 + col_loc[2] - diag[1][col_loc[1]][1] - 1) mod 4 + 1;

    return([diag[2][col_loc[1]], pos]);

end;

findLabelForEdge := function(edge, els)
    local i;
    # find label corresponding to edge
    for i in [1..Length(els)] do
        if Set(edge) = Set(els[i][1]) then
            return(els[i][2]);
        fi;
    od;
end;


findAllEdgeLabels:=function(node)
    local grp, nGrp, graphs, i, eLabel;

    verbose := false;
    createColumns();
    verbose := true;

    grp := findGraphForNode(node);
    nGrp := Length(grp[1]);

    graphs := [];
    Print("nGraph = ", nGrp, "\n");
    for i in [1..nGrp] do  # for each edge
        eLabel := findEdgeLabel(grp[1][i], grp, columns);

        # if node is in rank 1
        if node <= Length(R) + 1 then
            Add(graphs, [grp[1][i], eLabel{[1,2]}]);
        else
            if eLabel[2] mod 2 = 0 then  # even
                Add(graphs, [grp[1][i], [eLabel[1], 6]]);
```

```
                else                            # odd
                    Add(graphs, [grp[1][i], [eLabel[1], 5]]);
                fi;
            fi;

            Print(".\c");
        od;
        Print("\n");

        return(graphs);
    end;


    simpleLoopToEdgeLabels := function(loop, els)
        local nEdges, lbl, edge, i;

        nEdges := Length(loop);

        lbl := [];
        for i in [1..(nEdges-1)] do
            edge := loop{[i, i+1]};
            Add(lbl, findLabelForEdge(edge, els));
        od;

        return (lbl);
    end;


    simpleLoopsToEdgeLables := function(loops, els)
        local nLoops, EL, i;

        nLoops := Length(loops);

        EL := [];
        for i in [1..nLoops] do
            Add(EL, simpleLoopToEdgeLabels(loops[i], els));
        od;

        return (EL);
    end;


    countEdgeLabels := function(el)
        local nEL, tetraedges, countEL, i;

        nEL := Length(el);

        # constant
        tetraedges := 6;

        countEL := List([1..nPatterns * tetraedges], i->0);
        for i in [1..nEL] do
            countEL[(el[i][1]-1)*tetraedges + el[i][2]]
               := countEL[(el[i][1]-1)*tetraedges + el[i][2]]+1;
        od;

        return countEL;
    end;


    countAllEdgeLabels := function(EL)
        local i, nEL, countELs;

        nEL := Length(EL);
```

```
    countELs := [];
    for i in [1..nEL] do
        Add(countELs, countEdgeLabels(EL[i]));
    od;

    return Set(countELs);
end;


findAllIneqes := function()
    local nodes, allCountELs, graph, loops, ELs, countELs, i,Patterns;

    nodes := nodesCheck();
    createColumns();
    createSpces();
    nPatterns:=Length(tetrahedra);

    Print(" There are ", nPatterns," simplices.\n");
    Print(" The simplices are\n ",tetrahedra,".\n\n");

    Print(" There Are ",Length(nodes)," nodes to check which are ",nodes,".\n\n");

    allCountELs := [];
    for i in [1..Length(nodes)] do
        graph := findGraphForNode(nodes[i]);
        loops := findAllSimpleClosedLoopsForNode(nodes[i]);
        ELs := simpleLoopsToEdgeLables(loops, findAllEdgeLabels(nodes[i]));
        countELs := countAllEdgeLabels(ELs);
        allCountELs := UnionSet(countELs, allCountELs);
    od;

    return allCountELs;
end;


# compareList compares two elements r and s
# and gives true when r[i] is not more than s[i] for all i.
# subloop of elimRedundant.
# that is, s is redundant in our case.
compareList := function(r,s)
    local i;

    for i in [1..Length(r)] do
        if r[i]>s[i] then
            return false;
        fi;
    od;
    return true;
end;


# elimRedundant gets rid of redundant elements from L using compareList.
# subloop of simplifiedIneqes.
elimRedundant := function(L)
    local simplified, nL,flag, i, j;

    simplified :=[];
    nL := Length(L);
    flag := List([1..nL],i->0);

    for i in [1..nL] do
        for j in [1..nL] do
            if  (not i=j) and compareList(L[j],L[i]) then
                if not L[i]=L[j] then
```

```
                    flag[i] :=j;
                    break;
                elif L[i]=L[j] then
                    if i>j then
                        flag[i] := j;
                        break;
                    fi;
                fi;
            fi;
        od;
    od;
    for i in [1..nL] do
        if flag[i] =0 then
            Add(simplified,L[i]);
        fi;
    od;
    return ([flag,simplified]);
end;

# simplifiedIneqes returns inequations after eliminate redundants.
simplifiedIneqes := function()
    local ineq;
    ineq := elimRedundant(findAllIneqes());
    Print("We have ",Length(ineq[2])," inequalities.\n\n");
    return ineq;
end;

# numEntries produces a sequence in which each entry indicates
# the number of nonzero entries of corresponding entry of given list.
# subloop of sortList
numEntries := function(L)
    local i,j,ans;

    ans := List([1..Length(L)],i->0);
    for i in [1..Length(L)] do
        for j in L[i] do
            if not j=0 then
                ans[i] := ans[i]+1;
            fi;
        od;
    od;
    return ans;
end;


# sortList produces
#   1)the number of elements with each number of nonzero entries,
#   2) those elements sorted by the number of nonzero entries.
# subloop of sortIneqes.
sortList := function(L,numEntries)
    local i,j, sortLength,sortNum,sortL;

    sortLength := Maximum(numEntries);
    sortNum := List([1..sortLength],i->[0,0]);
    sortL := List([1..sortLength],i->[]);

    for i in [1..sortLength] do
        sortNum[i][1] :=i;
    od;
    for i in [1..Length(L)] do
        for j in [1..sortLength] do
            if numEntries[i]=j then
```

```
                sortNum[j][2] := sortNum[j][2]+1;
                Add(sortL[j],L[i]);
            fi;
        od;
    od;

    return([sortNum,sortL]);
end;


# sortIneqes sorts inequations up to the number of nonzero entries.
sortIneqes := function()
    local ineqes;
    ineqes := simplifiedIneqes()[2];
    return(sortList(ineqes,numEntries(ineqes)));
end;



# dihedralAngles generates a sequence of two six-tuples which indicates
# edges for the angle. subloop for simplifiedDihAng.
dihedralAngles := function(tetra)
    local t, dihAng, i ;

    t := tetra;
    dihAng := List([1..6],i->[]);

    # this is the same way each tetrahedron counted. not same to Jon's way.
    Add(dihAng[1],[t[1],t[2],t[3],t[4],t[5],t[6]]);
    Add(dihAng[2],[t[2],t[3],t[4],t[1],t[6],t[5]]);
    Add(dihAng[3],[t[3],t[4],t[1],t[2],t[5],t[6]]);
    Add(dihAng[4],[t[4],t[1],t[2],t[3],t[6],t[5]]);
    Add(dihAng[5],[t[5],t[3],t[6],t[1],t[4],t[2]]);
    Add(dihAng[6],[t[6],t[1],t[5],t[3],t[4],t[2]]);

    # edges for the reverse direction  which produces same angle.
    for i in [1..6] do
        Add(dihAng[i],
           [dihAng[i][1][1],dihAng[i][1][4],dihAng[i][1][3],
            dihAng[i][1][2],dihAng[i][1][6],dihAng[i][1][5]]);
    od;

    return dihAng;
end;

# simplifiedDihAng is to find same dihedral angles by comparing
# the surounding edges obtained from dihedralAngles.
simplifiedDihAng := function(dihAng)
    local modifiedDihAng, simplifiedDihAng,
          player,index, rest,flag,
          i;

    modifiedDihAng:=[];

    for i in [1..6] do
        Add(modifiedDihAng,Set(dihAng[i]));
    od;

    simplifiedDihAng := List([1..6],i->0);
    simplifiedDihAng[1]:=1;
    player := 1; # the standard one for matching
    index := 1;
    rest := [2,3,4,5,6]; # not matched yet
```

```
    while  Length(rest)>0 do
        flag :=[];
        for i in rest do
            if modifiedDihAng[i]=modifiedDihAng[player] then
                simplifiedDihAng[i] := index;
                Add(flag,i);
            fi;
        od;
        rest := Difference(rest,flag);
        index := index+1;
        if Length(rest)>0 then
            player := Minimum(rest);
        fi;
    od;

    return simplifiedDihAng;
end;


# dihAngForGp produces a sequence of dihedral angles of tetrahedra
# for the group using tetrahedra(tetrahedra is generated in createSpces)
dihAngForGp := function()
    local tetra, n, dihAngForGp, ang, max,
          i,j;

    tetra := tetrahedra;
    n := Length(tetra);
    dihAngForGp := List([1..6*n],i->0);

    for i in [1..n] do
        max := Maximum(dihAngForGp);
        for j in [1..6] do
            ang := simplifiedDihAng(dihedralAngles(tetra[i]));
            dihAngForGp[6*(i-1)+j] := ang[j]+max;
        od;
    od;

    Print("The dihedral angles for the complex are  ",dihAngForGp," .\n\n");
    return dihAngForGp;
end;


reducedIneq := function(ineq, dihAngForGp)
    local lengIneq, numAngles, reducedIneq,
          i, j;

    lengIneq := Length(ineq);
    numAngles := Maximum(dihAngForGp);
    reducedIneq := List([1..numAngles],i->0);

    for i in [1..numAngles] do
        for j in [1..lengIneq] do
            if dihAngForGp[j] = i then
                reducedIneq[i] := reducedIneq[i]+ineq[j];
            fi;
        od;
    od;

    return reducedIneq;
end;


reducedIneqes := function()
    local ineqes, numIneq, reducedIneqes, dihAngForTheGp, i;
```

```
    ineqes := simplifiedIneqes()[2];
    numIneq := Length(ineqes);
    reducedIneqes := [];
    dihAngForTheGp := dihAngForGp();

    for i in [1..numIneq] do
        Add(reducedIneqes, reducedIneq(ineqes[i],dihAngForTheGp));
    od;

    return reducedIneqes;
end;


simplifiedReducedIneqes := function()
    local ineq;
    ineq := elimRedundant(reducedIneqes());
    Print("We have ",Length(ineq[2])," inequalities.\n\n");
    return ineq;
end;
```

B. Matlab code

```
function [v, theta] = ineqTest(m)

%
% a = m(1), h = m(2)
%
% n <- m
% m(1) = a = n(1), m(2) = 9*a - 6*h = 9*n(1) - 6*n(2),
% m(3) = 4*a - 2*h = 4*n(1) - 2*n(2), m(4) = h = n(2);

% mapping
n = m;
m(1) = n(1);  m(2) = 9*n(1) - 6*n(2);
m(3) = 4*n(1) - 2*n(2);  m(4) = n(2);

confMatrix = [ ...
    1 1 1 1 4 4; ...
    4 1 4 1 1 1; ...
    1 1 1 1 3 4; ...
    3 1 4 1 1 1; ...
    4 1 3 1 1 1; ...
    1 1 1 2 3 3; ...
    1 1 2 1 3 3; ...
    2 1 1 1 3 3; ...
    3 1 3 1 2 1];

for i=1:size(confMatrix,1)
    m1 = m(confMatrix(i,:));

    j = (m1(1)+m1(5)-m1(2))/2;
    l = (m1(4)+m1(5)-m1(3))/2;
    k = (m1(1)+m1(4)-m1(6))/2;

    c1 = sqrt((m1(1)*m1(5)-j^2)/(m1(1)*m1(5)));
    c2 = sqrt((m1(1)*m1(4)-k^2)/(m1(1)*m1(4)));
    c3 = l/sqrt(m1(5)*m1(4)) - j/sqrt(m1(1)*m1(5))*k/sqrt(m1(1)*m1(4));

    theta(i) = acos(c3/(c1*c2));
end

%
% theta = alpha_A, beta A,
%         alpha_B, beta Bf, beta Bh,
%         alpha Ca, alpha Cb, gamma C, beta C
%

v(1) = theta(9) - pi/2;
v(2) = theta(8) - pi/3;
v(3) = theta(7) + theta(6)*2 - pi;
v(4) = theta(5) - pi/3;
v(5) = theta(4) + theta(9) - pi;
v(6) = theta(6) + 2*theta(3) - pi;
v(7) = 3*theta(5) + theta(2) - 2*pi;
v(8) = theta(2) + theta(5) - pi;
v(9) = theta(1) + 2*theta(3) + 2*theta(6) + theta(7)-2*pi;
v(10) = theta(1) + 4*theta(3) + theta(7)-2*pi;
v(11) = theta(1) + 2*theta(3) - pi;

if nargout == 0
     fprintf('%+.16f\n', v(1:11));
end;
```

```
function [v, theta] = ineqTest(m)

%
% new :
%   n <- m
%   m(1) = a = 3*n(2)/4 , m(2) = n(1) = b,
%   m(3) = n(2) = f;
%

n = m;
m(1) = 3*n(2)/4;
m(2) = n(1);
m(3) = n(2);

confMatrix = [ ...
    1 1 1 1 3 3; ...
    3 1 3 1 1 1; ...
    1 1 1 2 3 3; ...
    1 1 2 1 3 3; ...
    2 1 1 1 3 3; ...
    3 1 3 1 2 1; ...
    1 1 2 2 3 3; ...
    2 2 1 1 3 3; ...
    3 2 3 1 2 1; ...
    3 1 3 2 2 1; ...
    1 2 1 2 3 3; ...
    2 1 2 1 3 3; ...
    3 1 3 1 2 2];


for i=1:size(confMatrix,1)
    m1 = m(confMatrix(i,:));

    j = (m1(1)+m1(5)-m1(2))/2;
    l = (m1(4)+m1(5)-m1(3))/2;
    k = (m1(1)+m1(4)-m1(6))/2;


    c1 = sqrt((m1(1)*m1(5)-j^2)/(m1(1)*m1(5)));
    c2 = sqrt((m1(1)*m1(4)-k^2)/(m1(1)*m1(4)));
    c3 = l/sqrt(m1(5)*m1(4)) - j/sqrt(m1(1)*m1(5))*k/sqrt(m1(1)*m1(4));

    theta(i) = acos(c3/(c1*c2));
end

%
%
% For F4
% theta = alpha A, beta A,
%         alpha Ba, alpha Bb, gamma B, beta B,
%         alpha C, gamma C, beta Ca, beta Cb,
%         alpha C, gamma D, beta D.
%
%


% ineq 27 = v(1) : alpha Bb is at least pi/3
%                ---> 3*f less than or equal to 4*b
% ineq 24 = v(2) : alpha Ba is at least pi/3
% ineq 5  = v(3) : beta B is at least pi/2
%                ---> 4*b less than or equal to 3*f

v(1) = theta(1)*4 + theta(4)*2 - 2*pi;
v(2) = theta(1)*4 + theta(3)*2 - 2*pi;
```

```
    v(3) = theta(6)*4 - 2*pi;

    if nargout == 0
        fprintf('%+.16f\n', v(1:3));

70      fprintf('\n');
    end;
```

VITA

Woonjung Choi was born in Pusan, Korea in 1973. She received her B.S. degree in 1997 in the Department of Mathemetics, Pusan National University, Korea and she received her M.S. degree in 1999 in the same department. Afterwards, she started her Ph.D. in the Department of Mathematics, Texas A&M University. Her current research interests include geometric group theory and combinatorics. She can be reached at the following address: Department of Mathematics, Texas A&M University, College Station, TX 77843-3368.